# Contributing

Thanks for helping us to make KSQL even better!

If you have any questions about how to contribute, either [create a GH issue](https://github.com/confluentinc/ksql/issues) or ask your question in the #ksql channel in our public [Confluent Community Slack](https://slackpass.io/confluentcommunity) (account registration is free and self-service).

## Developing KSQL

### Building and running KSQL locally

To build and run KSQL locally, run the following commands:

```shell

$ mvn clean package -DskipTests

$ ./bin/ksql-server-start -daemon config/ksql-server.properties

$ ./bin/ksql

```

This will start the KSQL server in the background and the KSQL CLI in the

foreground. Check the `logs` folder for the log files that the server writes

including any errors.

If you would rather have the KSQL server logs spool to the console, then

drop the `-daemon` switch, and start the CLI in a second console.

### Testing changes locally

To build and test changes locally, run the following commands:

```shell

$ mvn verify

```

### Testing docker image

See comments at the top of the [docker compose file in the root of the project](docker-compose.yml) for instructions

on how to build and run docker images.

## How to Contribute

### Reporting Bugs and Issues

Report bugs and issues by creating a [new GitHub issue](https://github.com/confluentinc/ksql/issues).

Prior to creating an issue, please search through existing issues so that you are not creating duplicate ones.

### Guidelines for Contributing Code, Examples, Documentation

Any change to the public API of ksqlDB, especially the SQL syntax, requires a \*\*K\*\*sq\*\*l\*\*DB

\*\*i\*\*mprovement \*\*p\*\*roposal (KLIP) to be raised and approved.

See the [KLIP Readme](design-proposals/README.md) for more info.

Code changes are submitted via a pull request (PR). When submitting a PR use the following guidelines:

\* Follow the style guide below

\* Add/update documentation appropriately for the change you are making. For more information, see the [docs readme](docs/readme.md).

\* Non-trivial changes should include unit tests covering the new functionality and potentially [function tests](ksql-engine/src/test/resources/query-validation-tests/README.md).

\* All SQL syntax changes and enhancements should come with appropriate [function tests](ksql-engine/src/test/resources/query-validation-tests/README.md).

\* Bug fixes should include a unit test or integration test potentially [function tests](ksql-engine/src/test/resources/query-validation-tests/README.md) proving the issue is fixed.

\* Try to keep pull requests short and submit separate ones for unrelated features.

\* Keep formatting changes in separate commits to make code reviews easier and distinguish them from actual code changes.

#### Code Style

The project uses [GoogleStyle](https://google.github.io/styleguide/javaguide.html) code formatting.

You can install this code style into your IDE to make things more automatic:

\* [IntelliJ code style xml file](https://github.com/google/styleguide/blob/gh-pages/intellij-java-google-style.xml)

\* [Eclipse code style xml file](https://github.com/google/styleguide/blob/gh-pages/eclipse-java-google-style.xml)

The project also uses immutable types where ever possible. If adding new type(s), ideally make them immutable.

#### Static code analysis

The project build runs checkstyle and findbugs as part of the build.

You can set up IntelliJ for CheckStyle. First install the CheckStyle IDEA plugin, then:

IntelliJ->Preferences?Other Settings?CheckStyle

- Add a new configurations file using the '+' button:

Description: Confluent Checks

URL: https://raw.githubusercontent.com/confluentinc/common/master/build-tools/src/main/resources/checkstyle/checkstyle.xml

Ignore invalid certs: true

- (Optional) Make the new configuration active.

- Highlight the newly added 'Confluent Checks' and click the edit button (pencil icon).

- Set properties to match the `checkstyle/checkstyle.properties` file in the repo.

'Confluent Checks' will now be available in the CheckStyle tool window in the IDE and will auto-highlight issues in the code editor.

#### Commit messages

The project uses [Conventional Commits][https://www.conventionalcommits.org/en/v1.0.0-beta.4/] for commit messages

in order to aid in automatic generation of changelogs. As described in the Conventional Commmits specification,

commit messages should be of the form:

<type>[optional scope]: <description>

[optional body]

[optional footer]

where the `type` is one of

\* "fix": for bug fixes

\* "feat": for new features

\* "refactor": for refactors

\* "test": for test-only changes

\* "docs": for docs-only changes

\* "revert": for reverting other changes

\* "perf", "style", "build", "ci", or "chore": as described in the [Angular specification][https://github.com/angular/angular/blob/22b96b9/CONTRIBUTING.md#type] for Conventional Commits.

The (optional) scope is a noun describing the section of the codebase affected by the change.

Examples that could make sense for KSQL include "parser", "analyzer", "rest server", "testing tool",

"cli", "processing log", and "metrics", to name a few.

The optional body and footer are for specifying additional information, such as linking to issues fixed by the commit

or drawing attention to [breaking changes](#breaking-changes).

##### Breaking changes

A commit is a "breaking change" if users should expect different behavior from an existing workflow

as a result of the change. Examples of breaking changes include deprecation of existing configs or APIs,

changing default behavior of existing configs or query semantics, or the renaming of exposed JMX metrics.

Breaking changes must be called out in commit messages, PR descriptions, and upgrade notes:

\* Commit messages for breaking changes must include a line (in the optional body or footer)

starting with "BREAKING CHANGE: " followed by an explanation of what the breaking change was.

For example,

feat: inherit num partitions and replicas from source topic in CSAS/CTAS

BREAKING CHANGE: `CREATE \* AS SELECT` queries now create sink topics with partition

and replica count equal to those of the source, rather than using values from the properties

`ksql.sink.partitions` and `ksql.sink.replicas`. These properties are now deprecated.

\* The breaking change should similarly be called out in the PR description.

This description will be copied into the body of the final commit merged into the repo,

and picked up by our automatic changelog generation accordingly.

\* [Upgrade notes][https://github.com/confluentinc/ksql/blob/master/docs/installation/upgrading.rst]

should also be updated as part of the same PR.

##### Commitlint

This project has [commitlint][https://github.com/conventional-changelog/commitlint] configured

to ensure that commit messages are of the expected format.

To enable commitlint, simply run `npm install` from the root directory of the KSQL repo

(after [installing `npm`][https://www.npmjs.com/get-npm].)

Once enabled, commitlint will reject commits with improperly formatted commit messages.

### GitHub Workflow

1. Fork the `confluentinc/ksql` repository into your GitHub account: https://github.com/confluentinc/ksql/fork.

2. Clone your fork of the GitHub repository, replacing `<username>` with your GitHub username.

Use ssh (recommended):

```bash

git clone git@github.com:<username>/ksql.git

```

Or https:

```bash

git clone https://github.com/<username>/ksql.git

```

3. Add a remote to keep up with upstream changes.

```bash

git remote add upstream https://github.com/confluentinc/ksql.git

```

If you already have a copy, fetch upstream changes.

```bash

git fetch upstream

```

4. Create a feature branch to work in.

```bash

git checkout -b feature-xxx remotes/upstream/master

```

5. Work in your feature branch.

```bash

git commit -a

```

6. Periodically rebase your changes

```bash

git pull --rebase

```

7. When done, combine ("squash") related commits into a single one

```bash

git rebase -i upstream/master

```

This will open your editor and allow you to re-order commits and merge them:

- Re-order the lines to change commit order (to the extent possible without creating conflicts)

- Prefix commits using `s` (squash) or `f` (fixup) to merge extraneous commits.

8. Submit a pull-request

```bash

git push origin feature-xxx

```

Go to your fork main page

```bash

https://github.com/<username>/ksql

```

If you recently pushed your changes GitHub will automatically pop up a `Compare & pull request` button for any branches you recently pushed to. If you click that button it will automatically offer you to submit your pull-request to the `confluentinc/ksql` repository.

- Give your pull-request a meaningful title that conforms to the Conventional Commits specification

as described [above](#commit-messages) for commit messages.

You'll know your title is properly formatted once the `Semantic Pull Request` GitHub check

transitions from a status of "pending" to "passed".

- In the description, explain your changes and the problem they are solving. Be sure to also call out

any breaking changes as described [above](#breaking-changes).

9. Addressing code review comments

Repeat steps 5. through 7. to address any code review comments and rebase your changes if necessary.

Push your updated changes to update the pull request

```bash

git push origin [--force] feature-xxx

```

`--force` may be necessary to overwrite your existing pull request in case your

commit history was changed when performing the rebase.

Note: Be careful when using `--force` since you may lose data if you are not careful.

```bash

git push origin --force feature-xxx

```