# **Contributing**

We would ❤️ you to contribute to Appwrite and help make it better! We want contributing to Appwrite to be fun, enjoyable, and educational for anyone and everyone. All contributions are welcome, including issues and new docs, as well as updates and tweaks, blog posts, workshops, and more.

## **How to Start?**

If you are worried or don’t know where to start, check out the next section that explains what kind of help we could use and where you can get involved. You can reach out with questions to [Eldad Fux (@eldadfux)](https://twitter.com/eldadfux) or [@appwrite](https://twitter.com/appwrite) on Twitter or to anyone from the [Appwrite team on Discord](https://discord.gg/GSeTUeA). You can also submit an issue, and a maintainer can guide you!

## **Code of Conduct**

Help us keep Appwrite open and inclusive. Please read and follow our [Code of Conduct](https://github.com/appwrite/appwrite/blob/master/https:/github.com/appwrite/.github/blob/main/CODE_OF_CONDUCT.md).

## **Submit a Pull Request 🚀**

Branch naming convention is as follows

TYPE-ISSUE\_ID-DESCRIPTION

example:

doc-548-submit-a-pull-request-section-to-contribution-guide

When TYPE can be:

* feat - a new feature
* doc - documentation only changes
* cicd - changes related to CI/CD system
* fix - a bug fix
* refactor - code change that neither fixes a bug nor adds a feature

All PRs must include a commit message with the description of the changes made!

For the initial start, fork the project and use git clone command to download the repository to your computer. A standard procedure for working on an issue would be to:

1. git pull, before creating a new branch, pull the changes from upstream. Your master needs to be up to date.

$ git pull

1. Create a new branch from master like: doc-548-submit-a-pull-request-section-to-contribution-guide.

$ git checkout -b [name\_of\_your\_new\_branch]

1. Work - commit - repeat (be sure to be in your branch).
2. Before you push your changes, make sure your code follows the PSR12 coding standards, which is the standard that Appwrite currently follows. You can easily do this by running the formatter.

composer format <your file path>

Now, go a step further by running the linter using the following command to manually fix the issues the formatter wasn't able to.

composer lint <your file path>

This will give you a list of errors to rectify. If there is an instance in which you need more information on the errors being displayed, you can pass in additional command line arguments. More lists of available arguments can be found [here](https://github.com/squizlabs/PHP_CodeSniffer/wiki/Usage). A very useful command line argument is --report=diff. This will give you the expected changes by the linter for easy fixing of formatting issues.

composer lint --report=diff <your file path>

1. Push changes to GitHub.

$ git push origin [name\_of\_your\_new\_branch]

1. Submit your changes for review If you go to your repository on GitHub, you'll see a Compare & pull request button. Click on that button.
2. Start a Pull Request Now submit the pull request and click on Create pull request.
3. Get a code review approval/reject.
4. After approval, merge your PR.
5. GitHub will automatically delete the branch after the merge is done. (they can still be restored).

## **Setup From Source**

To set up a working development environment, just fork the project git repository and install the backend and frontend dependencies using the proper package manager and create run the docker-compose stack.

If you just want to install Appwrite for day-to-day usage and not as a code maintainer, use this [installation guide](https://github.com/appwrite/appwrite#installation).

Please note that these instructions are for setting a functional dev environment. If you want to set up an Appwrite instance to integrate into your app, you should probably install Appwrite using the instructions in the [getting started guide](https://appwrite.io/docs/getting-started-for-web) or in the main [README](https://github.com/appwrite/appwrite/blob/master/README.md) file.

git clone git@github.com:[YOUR\_FORK\_HERE]/appwrite.git

cd appwrite

docker compose build

docker compose up -d

### **Code Autocompletion**

To get proper autocompletion for all the different functions and classes in the codebase, you'll need to install Appwrite dependencies on your local machine. You can easily do that with PHP's package manager, [Composer](https://getcomposer.org/). If you don't have Composer installed, you can use the Docker Hub image to get the same result:

docker run --rm --interactive --tty \

--volume $PWD:/app \

composer update --ignore-platform-reqs --optimize-autoloader --no-plugins --no-scripts --prefer-dist

### **User Interface**

Appwrite uses an internal micro-framework called Litespeed.js to build simple UI components in vanilla JS and [less](http://lesscss.org/) for compiling CSS code. To apply any of your changes to the UI, use the gulp build or gulp less commands, and restart the Appwrite main container to load the new static files to memory using docker compose restart appwrite.

### **Get Started**

After finishing the installation process, you can start writing and editing code.

#### **Advanced Topics**

We love to create issues that are good for beginners and label them as good first issue or hacktoberfest, but some more advanced topics might require extra knowledge. Below is a list of links you can use to learn about the more advanced topics that will help you master the Appwrite codebase.

##### **Tools and Libs**

* [Docker](https://www.docker.com/get-started)
* [PHP FIG](https://www.php-fig.org/) - [PSR-12](https://www.php-fig.org/psr/psr-12/)
* [PHP Swoole](https://www.swoole.co.uk/)

Learn more at our [Technology Stack](https://github.com/appwrite/appwrite/blob/master/CONTRIBUTING.md#technology-stack) section.

##### **Network and Protocols**

* [OSI Model](https://en.wikipedia.org/wiki/OSI_model)
* [TCP vs UDP](https://www.guru99.com/tcp-vs-udp-understanding-the-difference.html#:~:text=TCP%20is%20a%20connection%2Doriented,speed%20of%20UDP%20is%20faster&text=TCP%20does%20error%20checking%20and,but%20it%20discards%20erroneous%20packets.)
* [HTTP](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol)
* [REST API](https://en.wikipedia.org/wiki/Representational_state_transfer)
* [GraphQL](https://en.wikipedia.org/wiki/GraphQL)
* [gRPC](https://en.wikipedia.org/wiki/GRPC)

##### **Architecture**

* [Microservices vs Monolithic](https://www.mulesoft.com/resources/api/microservices-vs-monolithic#:~:text=Microservices%20architecture%20vs%20monolithic%20architecture&text=A%20monolithic%20application%20is%20built%20as%20a%20single%20unit.&text=To%20make%20any%20alterations%20to,formally%20with%20business%2Doriented%20APIs.)
* [MVVM](https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel) - Appwrite console architecture

##### **Security**

* [Appwrite Auth and ACL](https://github.com/appwrite/appwrite/blob/master/docs/specs/authentication.drawio.svg)
* [OAuth](https://en.wikipedia.org/wiki/OAuth)
* [Encryption](https://medium.com/searchencrypt/what-is-encryption-how-does-it-work-e8f20e340537#:~:text=Encryption%20is%20a%20process%20that,%2C%20or%20decrypt%2C%20the%20information.)
* [Hashing](https://searchsqlserver.techtarget.com/definition/hashing#:~:text=Hashing%20is%20the%20transformation%20of,it%20using%20the%20original%20value.)

## **Architecture**

Appwrite's current structure is a combination of both [Monolithic](https://en.wikipedia.org/wiki/Monolithic_application) and [Microservice](https://en.wikipedia.org/wiki/Microservices) architectures, but as we grow, our goal is to only use microservices.

## Appwrite

### **File Structure**

.

├── app # Main application

│ ├── config # Config files

│ ├── controllers # API & dashboard controllers

│ │ ├── api

│ │ ├── shared

│ │ └── web

│ ├── db # DB schemas

│ ├── sdks # SDKs generated copies (used for generating code examples)

│ ├── tasks # Server CLI commands

│ ├── views # HTML server-side templates

│ └── workers # Background workers

├── bin # Server executables (tasks & workers)

├── docker # Docker related resources and configs

├── docs # Docs and tutorials

│ ├── examples

│ ├── references

│ ├── services

│ ├── specs

│ └── tutorials

├── public # Public files

│ ├── dist

│ ├── fonts

│ ├── images

│ ├── scripts

│ └── styles

├── src # Supporting libraries (each lib has one role, common libs are released as individual projects)

│ └── Appwrite

│ ├── Auth

│ ├── Database

│ ├── Detector

│ ├── Docker

│ ├── Event

│ ├── Extend

│ ├── Migration

│ ├── Network

│ ├── OpenSSL

│ ├── Realtime

│ ├── Resque

│ ├── Specification

│ ├── Task

│ ├── Template

│ ├── URL

│ └── Utopia

└── tests # End to end & unit tests

├── e2e

├── resources

└── unit

### **The Monolithic Part**

Appwrite's main API container is designed as a monolithic app. This is a decision we made to allow us to develop the project faster while still being a very small team.

Although the Appwrite API is a monolithic app, it has a very clear separation of concern as each internal service or worker is separated by its container, which allows us to start breaking services for better maintenance and scalability as we grow.

### **The Microservice Part**

Each container in Appwrite is a microservice on its own. Each service is an independent process that can scale without regard to any of the other services.

Currently, all the Appwrite microservices are intended to communicate using the TCP protocol over a private network. With the exception of the public port 80 and 443, which, by default, are used to expose the Appwrite HTTP API, you should be careful not to expose any services to the public-facing network.

## **Ports**

Appwrite dev version uses ports 80 and 443 as an entry point to the Appwrite API and console. We also expose multiple ports in the range of 9500–9504 for debugging some of the Appwrite containers on dev mode. If you have any conflicts with the ports running on your system, you can easily replace them by editing Appwrite's docker-compose.yml file and executing docker compose up -d command.

## **Technology Stack**

To start helping us to improve the Appwrite server by submitting code, prior knowledge of Appwrite's technology stack can help you get started.

Appwrite stack is a combination of a variety of open-source technologies and tools. Appwrite backend API is written primarily with PHP version 7 and above, on top of the [Utopia PHP framework](https://github.com/utopia-php/framework). The Appwrite frontend is built with tools like gulp, less, and [litespeed.js](https://github.com/litespeed-js). We use Docker as the container technology to package the Appwrite server for easy on-cloud, on-premise, or on-localhost integration.

### **Other Technologies**

* Redis - for managing cache and in-memory data (currently, we do not use Redis for persistent data).
* MariaDB - for database storage and queries.
* InfluxDB - for managing stats and time-series based data.
* Statsd - for sending data over UDP protocol (using Telegraf).
* ClamAV - for validating and scanning storage files.
* Imagemagick - for manipulating and managing image media files.
* Webp - for better compression of images on supporting clients.
* SMTP - for sending email messages and alerts.
* Resque - for managing data queues and scheduled tasks over a Redis server.

## **Package Managers**

Appwrite uses a package manager for managing code dependencies for both backend and frontend development. We try our best to avoid creating any unnecessary dependencies, and any new dependency to the project is subject to a lead developer review and approval.

Many of Appwrite's internal modules are also used as dependencies to allow other Appwrite projects to reuse them and as a way to contribute them to the community.

Appwrite uses [PHP's Composer](https://getcomposer.org/) for managing dependencies on the server-side and [JS NPM](https://www.npmjs.com/) for managing dependencies on the frontend side.

## **Coding Standards**

Appwrite follows the [PHP-FIG standards](https://www.php-fig.org/). Currently, we use both PSR-0 and PSR-12 for coding standards and autoloading standards.

We use prettier for our JS coding standards and auto-formatting for our code.

## **Scalability, Speed, and Performance**

Appwrite is built to scale. Please keep in mind that the Appwrite stack can run in different environments and different scales.

Our intention is for Appwrite to be as easy to set up in a single localhost as it is to grow to a large environment with dozens and even hundreds of instances.

When contributing code, please take into account the following:

* Response Time
* Throughput
* Requests per Seconds
* Network Usage
* Memory Usage
* Browser Rendering
* Background Jobs
* Task Execution Time

## **Security and Privacy**

Security and privacy are extremely important to Appwrite, developers, and users alike. Make sure to follow the best industry standards and practices.

## **Dependencies**

Please avoid introducing new dependencies to Appwrite without consulting the team. New dependencies can be very helpful, but they also introduce new security and privacy issues, add complexity and impact the total docker image size.

Adding a new dependency should have vital value for the product with minimum possible risk.

## **Introducing New Features**

We would 💖 you to contribute to Appwrite, but we also want to ensure that Appwrite is as great as possible and loyal to its vision and mission statement 🙏.

For us to find the right balance, please open an issue explaining your ideas before introducing a new pull request.

This will allow the Appwrite community to sufficiently discuss the new feature value and how it fits with the product roadmap and vision.

This is also important for the Appwrite lead developers to be able to provide technical input and potentially a different emphasis regarding the feature design and architecture. Some bigger features might need to go through our [RFC process](https://github.com/appwrite/rfc).

## **Build**

To build a new version of the Appwrite server, all you need to do is run the build.sh file like this:

bash ./build.sh X.X.X

Before running the command, make sure you have proper write permissions to the Appwrite docker hub team.

Build for Multicore

docker buildx build --platform linux/amd64,linux/arm64,linux/arm/v6,linux/arm/v7,linux/arm64/v8,linux/ppc64le,linux/s390x -t appwrite/appwrite:dev --push .

Build Functions Runtimes

The Runtimes for all supported cloud functions (multicore builds) can be found at the [open-runtimes/open-runtimes](https://github.com/open-runtimes/open-runtimes) repository.

## **Generate SDK**

The following steps are used to generate a new console SDK:

1. Update the console spec file located at app/config/specs/swagger2-<version-number>.console.json using Appwrite Tasks. Run the php app/cli.php specs <version-number> normal command in a running appwrite/appwrite container.
2. Generate a new SDK using the command php app/cli.php sdks.
3. Change your working dir using cd app/sdks/console-web.
4. Build the new SDK npm run build.
5. Copy iife/sdk.js to appwrite.js.
6. Go back to the root of the project run npm run build.

## **Checklist for Releasing SDKs**

Things to remember when releasing SDKs:

* Update the Changelogs in docs/sdks (right now only Dart and Flutter are using these).
* Update GETTING\_STARTED.md in docs/sdks for each SDKs if any changes in the related APIs are in there.
* Update SDK versions as required on app/config/platforms.php.
* Generate SDKs using the command php app/cli.php sdks and follow the instructions.
* Release new tags on GitHub repository for each SDK.

## **Debug**

Appwrite uses [yasd](https://github.com/swoole/yasd) debugger, which can be made available during build of Appwrite. You can connect to the debugger using VS Code [PHP Debug](https://marketplace.visualstudio.com/items?itemName=felixfbecker.php-debug) extension; if you are in PHP Storm, no plugin is needed. Below are the settings required for the remote debugger connection.

First, you need to create an init file. Duplicate dev/yasd\_init.php.stub file and name it dev/yasd\_init.php and then change the IP address to your development machine's IP. Without the proper IP address, the debugger won't connect. You also need to set DEBUG build arg in appwrite service in docker-compose.yml file.

### **VS Code Launch Configuration**

{

"name": "Listen for Xdebug",

"type": "php",

"request": "launch",

"port": 9005,

"pathMappings": {

"/usr/src/code": "${workspaceRoot}"

}

}

### **PHPStorm Setup**

In settings, go to Languages & Frameworks > PHP > Debug. Under Xdebug, set the debug port to 9005 and enable to accept the external connections checkbox.

## **Tests**

To run all tests manually, use the Appwrite Docker CLI from your terminal:

docker compose exec appwrite test

To run unit tests use:

docker compose exec appwrite test /usr/src/code/tests/unit

To run end-2-end tests use:

docker compose exec appwrite test /usr/src/code/tests/e2e

To run end-2-end tests for a specific service use:

docker compose exec appwrite test /usr/src/code/tests/e2e/Services/[ServiceName]

## **Benchmarking**

You can use WRK Docker image to benchmark the server performance. Benchmarking is extremely useful when you want to compare how the server behaves before and after a change has been applied. Replace [APPWRITE\_HOSTNAME\_OR\_IP] with your Appwrite server hostname or IP. Note that localhost is not accessible from inside the WRK container.

Options:

-c, --connections <N> Connections to keep open

-d, --duration <T> Duration of test

-t, --threads <N> Number of threads to use

-s, --script <S> Load Lua script file

-H, --header <H> Add header to request

--latency Print latency statistics

--timeout <T> Socket/request timeout

-v, --version Print version details

docker run --rm skandyla/wrk -t3 -c100 -d30 https://[APPWRITE\_HOSTNAME\_OR\_IP]

## **Code Maintenance**

We use some automation tools to help us keep a healthy codebase.

Run Formatter:

# Run on all files

composer format

# Run on single file or folder

composer format <your file path>

Run Linter:

# Run on all files

composer lint

# Run on single file or folder

composer lint <your file path>

## **Tutorials**

From time to time, our team will add tutorials that will help contributors find their way in the Appwrite source code. Below is a list of currently available tutorials:

* [Adding Support for a New OAuth2 Provider](https://github.com/appwrite/appwrite/blob/master/docs/tutorials/add-oauth2-provider.md)
* [Appwrite Environment Variables](https://github.com/appwrite/appwrite/blob/master/docs/tutorials/environment-variables.md)
* [Running in Production](https://github.com/appwrite/appwrite/blob/master/docs/tutorials/running-in-production.md)
* [Adding Storage Adapter](https://github.com/appwrite/appwrite/blob/master/docs/tutorials/add-storage-adapter.md)

## **Other Ways to Help**

Pull requests are great, but there are many other ways you can help Appwrite.

### **Blogging & Speaking**

Blogging, speaking about, or creating tutorials about one of Appwrite’s many features are great ways to get the word out about Appwrite. Mention [@appwrite](https://twitter.com/appwrite) on Twitter and/or email team@appwrite.io so we can give pointers and tips and help you spread the word by promoting your content on the different Appwrite communication channels. Please add your blog posts and videos of talks to our [Awesome Appwrite](https://github.com/appwrite/awesome-appwrite) repo on GitHub.

### **Presenting at Meetups**

We encourage our contributors to present at meetups and conferences about your Appwrite projects. Your unique challenges and successes in building things with Appwrite can provide great speaking material. We’d love to review your talk abstract/CFP, so get in touch with us if you’d like some help!

### **Sending Feedback & Reporting Bugs**

Sending feedback is a great way for us to understand your different use cases of Appwrite better. If you had any issues, bugs, or want to share your experience, feel free to do so on our GitHub issues page or at our [Discord channel](https://discord.gg/GSeTUeA).

### **Submitting New Ideas**

If you think Appwrite could use a new feature, please open an issue on our GitHub repository, stating as much information as you have about your new idea and its implications. We would also use this issue to gather more information, get more feedback from the community, and have a proper discussion about the new feature.

### **Improving Documentation**

Submitting documentation updates, enhancements, designs, or bug fixes, as well as spelling or grammar fixes, is very much appreciated.

### **Helping Someone**

Please consider searching for Appwrite on Discord, GitHub, or StackOverflow and helping someone else who needs help. You can also help by teaching others how to contribute to Appwrite's repo!