Instalação Cal.dyi nova versão Cal

Link: https://github.com/calcom/cal.diy/tree/main git clone https://github.com/calcom/cal.diy.git
Em 24/08/2026

A ÚLTIMA VERSÃO CAL.COM FOI LANÇADA EM OUTUBRO/2025 COM A VERSÃO 5.8.7 DEPOIS PASSOU A SER CAL.DIY SEM RECURSOS ENTERPRISE.

Warning

Use at your own risk. Cal.diy is the open source community edition of Cal.com and it is intended for users who want to self-host their own Cal.diy instance. It is strictly recommended for personal, non-production use. Please review all installation and configuration steps carefully. Self-hosting requires advanced knowledge of server administration, database management, and securing sensitive data. Proceed only if you are comfortable with these responsibilities.

Tip

For any commercial and enterprise-ready scheduling infrastructure, use Cal.com, not Cal.diy; hosted by us or get invited to on-prem enterprise access here: https://cal.com/sales

Logo

Cal.diy

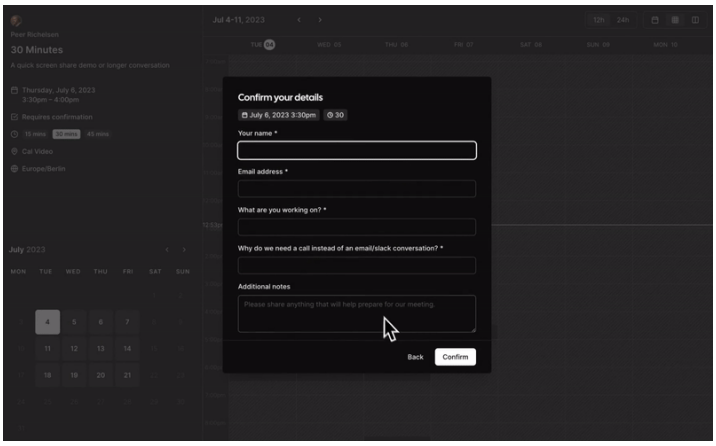
The community-driven, open-source scheduling platform.

GitHub

Issues · Contributing

License Github Stars Commits-per-month Docker Pulls
68747470733a2f2f696d672e736869656c64732e696f2f62616467652f48656c7025323057616e7465
68747470733a2f2f696d672e736869656c64732e696f2f62616467652f436f6e7472696275746f722253

About Cal.diy



booking-screen

Cal.diy is the community-driven, fully open-source scheduling platform — a fork of Cal.com with all enterprise/commercial code removed.

Cal.diy is **100% MIT-licensed** with no proprietary "Enterprise Edition" features. It's designed for individuals and self-hosters who want full control over their scheduling infrastructure without any commercial dependencies.

What's different from Cal.com?

- **No enterprise features** — Teams, Organizations, Insights, Workflows, SSO/SAML, and other EE-only features have been removed
- **No license key required** — Everything works out of the box, no Cal.com account or license needed
- **100% open source** — The entire codebase is licensed under MIT, no "Open Core" split
- **Community-maintained** — Contributions are welcome and go directly into this project (see CONTRIBUTING.md)

> **Note:** Cal.diy is a self-hosted project. There is no hosted/managed version. You run it on your own infrastructure.

Built With

- Next.js
- tRPC
- React.js
- Tailwind CSS
- Prisma.io
- Daily.co

Getting Started

To get a local copy up and running, please follow these simple steps.

Prerequisites

Here's what you need to run Cal.diy.

- Node.js (Version: >=18.x)
- PostgreSQL (Version: >=13.x)
- Yarn *(recommended)*

> If you want to enable any of the available integrations, you may want to obtain additional credentials for each one. More details on this can be found below under the integrations section.

Development

Setup

1. Clone the repo (or fork https://github.com/calcom/cal.diy/fork)

```
git clone https://github.com/calcom/cal.diy.git
```

> If you are on Windows, run the following command in Git Bash with admin privileges: `git clone -c core.symlinks=true https://github.com/calcom/cal.diy.git`

2. Go to the project folder

```
cd cal.diy
```

3. Install packages with yarn

```
yarn
```

4. Set up your `.env` file
   - Duplicate `.env.example` to `.env`
   - Use `openssl rand -base64 32` to generate a key and add it under `NEXTAUTH_SECRET` in the `.env` file.
   - Use `openssl rand -base64 24` to generate a key and add it under `CALENDSO_ENCRYPTION_KEY` in the `.env` file.

> **Windows users:** Replace the `packages/prisma/.env` symlink with a real copy to avoid a Prisma error (`unexpected character / in variable name`):
>
> ```
> # Git Bash / WSL
> rm packages/prisma/.env && cp .env packages/prisma/.env
> ```

5. Set up Node If your Node version does not meet the project's requirements as instructed by the docs, "nvm" (Node Version Manager) allows using Node at the version required by the project:

```
nvm use
```

You first might need to install the specific version and then use it:

```
nvm install && nvm use
```

You can install nvm from here.

Quick start with `yarn dx`

> - **Requires Docker and Docker Compose to be installed**
> - Will start a local Postgres instance with a few test users - the credentials will be logged in the console

```
yarn dx
```

Default credentials created:

Email	Password	Role
`free@example.com`	`free`	Free user
`pro@example.com`	`pro`	Pro user
`trial@example.com`	`trial`	Trial user
`admin@example.com`	`ADMINadmin2022!`	Admin user
`onboarding@example.com`	`onboarding`	Onboarding incomplete

You can use any of these credentials to sign in at http://localhost:3000

> **Tip**: To view the full list of seeded users and their details, run `yarn db-studio` and visit http://localhost:5555

Development tip

1. Add `export NODE_OPTIONS="--max-old-space-size=16384"` to your shell script to increase the memory limit for the node process. Alternatively, you can run this in your terminal before running the app. Replace 16384 with the amount of RAM you want to allocate to the node process.
2. Add `NEXT_PUBLIC_LOGGER_LEVEL={level}` to your .env file to control the logging verbosity for all tRPC queries and mutations.
   Where {level} can be one of the following:
   `0` for silly
   `1` for trace
   `2` for debug
   `3` for info
   `4` for warn
   `5` for error
   `6` for fatal
   When you set `NEXT_PUBLIC_LOGGER_LEVEL={level}` in your .env file, it enables logging at that level and higher. Here's how it works:

The logger will include all logs that are at the specified level or higher. For example: \

- If you set `NEXT_PUBLIC_LOGGER_LEVEL=2`, it will log from level 2 (debug) upwards, meaning levels 2 (debug), 3 (info), 4 (warn), 5 (error), and 6 (fatal) will be logged. \
- If you set `NEXT_PUBLIC_LOGGER_LEVEL=3`, it will log from level 3 (info) upwards, meaning levels 3 (info), 4 (warn), 5 (error), and 6 (fatal) will be logged, but level 2 (debug) and level 1 (trace) will be ignored. \

```
echo 'NEXT_PUBLIC_LOGGER_LEVEL=3' >> .env
```

for Logger level to be set at info, for example.

Gitpod Setup

1. Click the button below to open this project in Gitpod.
2. This will open a fully configured workspace in your browser with all the necessary dependencies already installed.

Open in Gitpod

Manual setup

1. Configure environment variables in the `.env` file. Replace `<user>`, `<pass>`, `<db-host>`, and `<db-port>` with their applicable values

   ```
   DATABASE_URL='postgresql://<user>:<pass>@<db-host>:<db-port>'
   ```

If you don't know how to configure the DATABASE_URL, then follow the steps here to create a quick local DB

1.
2.
3.
4. image
5.

1. If you don't want to create a local DB. Then you can also consider using services like railway.app, Northflank or render.
   - Setup postgres DB with railway.app
   - Setup postgres DB with Northflank
   - Setup postgres DB with render
2. Copy and paste your `DATABASE_URL` from `.env` to `.env.appStore`.

3. Set up the database using the Prisma schema (found in `packages/prisma/schema.prisma`)
   In a development environment, run:

   ```
   yarn workspace @calcom/prisma db-migrate
   ```

   In a production environment, run:

   ```
   yarn workspace @calcom/prisma db-deploy
   ```

Note for Windows/PowerShell users: If running the database deployment scripts fails with an error stating `Environment variable not found: DATABASE_DIRECT_URL`, Turbo might be failing to inject the root `.env` variables. You can bypass this by executing the commands directly from the prisma package directory in PowerShell:

```
cd packages/prisma
$env:DATABASE_URL="postgresql://postgres:YOUR_PASSWORD@localhost:5432/postgres"; $env:DATABASE_D
npx prisma db push
cd ../..
```

4. Run mailhog to view emails sent during development

   > ***NOTE:*** Required when `E2E_TEST_MAILHOG_ENABLED` is "1"

   ```
   docker pull mailhog/mailhog
   docker run -d -p 8025:8025 -p 1025:1025 mailhog/mailhog
   ```

5. Run (in development mode)

   ```
   yarn dev
   ```

Setting up your first user

Approach 1

1. Open Prisma Studio to look at or modify the database content:

   ```
   yarn db-studio
   ```

2. Click on the `User` model to add a new user record.

3. Fill out the fields `email`, `username`, `password`, and set `metadata` to empty `{}` (remembering to encrypt your password with BCrypt) and click `Save 1 Record` to create your first user.

   > New users are set on a `TRIAL` plan by default. You might want to adjust this behavior to your needs in the `packages/prisma/schema.prisma` file.

4. Open a browser to http://localhost:3000 and login with your just created, first user.

Approach 2

Seed the local db by running

```
cd packages/prisma
yarn db-seed
```

The above command will populate the local db with dummy users.

E2E-Testing

Be sure to set the environment variable `NEXTAUTH_URL` to the correct value. If you are running locally, as the documentation within `.env.example` mentions, the value should be `http://localhost:3000`.

```
# In a terminal just run:
yarn test-e2e

# To open the last HTML report run:
yarn playwright show-report test-results/reports/playwright-html-report
```

Resolving issues

E2E test browsers not installed

Run `npx playwright install` to download test browsers and resolve the error below when running `yarn test-e2e`:

```
Executable doesn't exist at /Users/alice/Library/Caches/ms-playwright/chromium-1048/chrome-mac/Chromium.app/Contents/MacOS/Chromium
```

Upgrading from earlier versions

1. Pull the current version:

   ```
   git pull
   ```

2. Check if dependencies got added/updated/removed

   ```
   yarn
   ```

3. Apply database migrations by running **one of** the following commands:
   In a development environment, run:

   ```
   yarn workspace @calcom/prisma db-migrate
   ```

   (This can clear your development database in some cases)
   In a production environment, run:

   ```
   yarn workspace @calcom/prisma db-deploy
   ```

4. Check for `.env` variables changes

   ```
   yarn predev
   ```

5. Start the server. In a development environment, just do:

   ```
   yarn dev
   ```

   For a production build, run for example:

   ```
   yarn build
   yarn start
   ```

6. Enjoy the new version.

Deployment

Docker

The Docker image can be found on DockerHub at https://hub.docker.com/r/calcom/cal.diy.

Note for ARM Users: Use the {version}-arm suffix for pulling images. Example: `docker pull calcom/cal.diy:v5.6.19-arm`.

Requirements

Make sure you have `docker` & `docker compose` installed on the server / system. Both are installed by most docker utilities, including Docker Desktop and Rancher Desktop.

Note: `docker compose` without the hyphen is now the primary method of using docker-compose, per the Docker documentation.

Running Cal.diy with Docker Compose

1. Clone the repository

   ```
   git clone --recursive https://github.com/calcom/cal.diy.git
   ```

2. Change into the directory

   ```
   cd cal.diy
   ```

3. Prepare your configuration: Rename `.env.example` to `.env` and then update `.env`

   ```
   cp .env.example .env
   ```

   Most configurations can be left as-is, but for configuration options see Important Run-time variables below.
   Required Secret Keys
   Before starting, you must generate secure values for `NEXTAUTH_SECRET` and `CALENDSO_ENCRYPTION_KEY`. Using the default `secret` placeholder in production is a security risk.
   Generate `NEXTAUTH_SECRET` (cookie encryption key):

   ```
   openssl rand -base64 32
   ```

   Generate `CALENDSO_ENCRYPTION_KEY` (must be 32 bytes for AES256):

   ```
   openssl rand -base64 24
   ```

Update your `.env` file with these values:

```
NEXTAUTH_SECRET=<your_generated_secret>
CALENDSO_ENCRYPTION_KEY=<your_generated_key>
```

Push Notifications (VAPID Keys) If you see an error like:

```
Error: No key set vapidDetails.publicKey
```

This means your environment variables for Web Push are missing. You must generate and set `NEXT_PUBLIC_VAPID_PUBLIC_KEY` and `VAPID_PRIVATE_KEY`.
Generate them with:

```
npx web-push generate-vapid-keys
```

Then update your `.env` file:

```
NEXT_PUBLIC_VAPID_PUBLIC_KEY=your_public_key_here
VAPID_PRIVATE_KEY=your_private_key_here
```

Do **not** commit real keys to `.env.example` — only placeholders.
Update the appropriate values in your .env file, then proceed.

4. (optional) Pre-Pull the images by running the following command:

```
docker compose pull
```

5. Start Cal.diy via docker compose
   To run the complete stack, which includes a local Postgres database, Cal.diy web app, and Prisma Studio:

```
docker compose up -d
```

To run Cal.diy web app and Prisma Studio against a remote database, ensure that DATABASE_URL is configured for an available database and run:

```
docker compose up -d calcom studio
```

To run only the Cal.diy web app, ensure that DATABASE_URL is configured for an available database and run:

```
docker compose up -d calcom
```

Note: to run in attached mode for debugging, remove `-d` from your desired run command.

6. Open a browser to http://localhost:3000, or your defined NEXT_PUBLIC_WEBAPP_URL. The first time you run Cal.diy, a setup wizard will initialize. Define your first user, and you're ready to go!
   Note for first-time setup (Calendar integration): During the setup wizard, you may encounter a "Connect your Calendar" step that appears to be required. If you do not wish to connect a calendar at this time, you can skip this step by navigating directly to the dashboard at `<NEXT_PUBLIC_WEBAPP_URL>/event-types`. Calendar integrations can be added later from the Settings > Integrations page.

Updating Cal.diy

1. Stop the Cal.diy stack
   ```
   docker compose down
   ```
2. Pull the latest changes
   ```
   docker compose pull
   ```
3. Update env vars as necessary.
4. Re-start the Cal.diy stack
   ```
   docker compose up -d
   ```

Building from source with Docker

1. Clone the repository
   ```
   git clone https://github.com/calcom/cal.diy.git
   ```
2. Change into the directory
   ```
   cd cal.diy
   ```
3. Rename `.env.example` to `.env` and then update `.env`
   For configuration options see Build-time variables below. Update the appropriate values in your .env file, then proceed.
4. Build the Cal.diy docker image:
   Note: Due to application configuration requirements, an available database is currently required during the build process.
   a) If hosting elsewhere, configure the `DATABASE_URL` in the .env file, and skip the next step

b) If a local or temporary database is required, start a local database via docker compose.

```
docker compose up -d database
```

5. Build Cal.diy via docker compose (DOCKER_BUILDKIT=0 must be provided to allow a network bridge to be used at build time. This requirement will be removed in the future)

```
DOCKER_BUILDKIT=0 docker compose build calcom
```

6. Start Cal.diy via docker compose
   To run the complete stack, which includes a local Postgres database, Cal.diy web app, and Prisma Studio:

```
docker compose up -d
```

   To run Cal.diy web app and Prisma Studio against a remote database, ensure that DATABASE_URL is configured for an available database and run:

```
docker compose up -d calcom studio
```

   To run only the Cal.diy web app, ensure that DATABASE_URL is configured for an available database and run:

```
docker compose up -d calcom
```

   Note: to run in attached mode for debugging, remove `-d` from your desired run command.

7. Open a browser to http://localhost:3000, or your defined NEXT_PUBLIC_WEBAPP_URL. The first time you run Cal.diy, a setup wizard will initialize. Define your first user, and you're ready to go!

Configuration

Important Run-time variables

These variables must also be provided at runtime

Variable	Description	Required	Default
DATABASE_URL	database url with credentials - if using a connection pooler, this setting should point there	required	`postgresql://unicorn_user:magical_password@database:5432/calendso`

Variable	Description	Required	Default
NEXT_PUBLIC_WEBAPP_URL	Base URL of the site. NOTE: if this value differs from the value used at build-time, there will be a slight delay during container start (to update the statically built files).	optional	`http://localhost:3000`
NEXTAUTH_URL	Location of the auth server. By default, this is the Cal.diy docker instance itself.	optional	`{NEXT_PUBLIC_WEBAPP_URL}/api/auth`
NEXTAUTH_SECRET	Cookie encryption key. Must match build variable. Generate with: `openssl rand -base64 32`	required	`secret`
CALENDSO_ENCRYPTION_KEY	Authentication encryption key (32 bytes for AES256). Must match build variable. Generate with: `openssl rand -base64 24`	required	`secret`

Build-time variables

If building the image yourself, these variables must be provided at the time of the docker build, and can be provided by updating the .env file. Currently, if you require changes to these variables, you must follow the instructions to build and publish your own image.

Variable	Description	Required	Default
DATABASE_URL	database url with credentials - if using a connection pooler, this setting should point there	required	`postgresql://unicorn_user:magical_password@database:5432/calendso`
MAX_OLD_SPACE_SIZE	Needed for Nodejs/NPM build options	required	4096
NEXTAUTH_SECRET	Cookie encryption key	required	`secret`
CALENDSO_ENCRYPTION_KEY	Authentication encryption key	required	`secret`
NEXT_PUBLIC_WEBAPP_URL	Base URL injected into static files	optional	`http://localhost:3000`
NEXT_PUBLIC_WEBSITE_TERMS_URL	custom URL for terms and conditions website	optional	
NEXT_PUBLIC_WEBSITE_PRIVACY_POLICY_URL	custom URL for privacy policy website	optional	

Variable	Description	Required	Default
CALCOM_TELEMETRY_DISABLED	Allow Cal.diy to collect anonymous usage data (set to `1` to disable)	optional	

Troubleshooting

SSL edge termination

If running behind a load balancer which handles SSL certificates, you will need to add the environmental variable `NODE_TLS_REJECT_UNAUTHORIZED=0` to prevent requests from being rejected. Only do this if you know what you are doing and trust the services/load-balancers directing traffic to your service.

Failed to commit changes: Invalid 'prisma.user.create()'

Certain versions may have trouble creating a user if the field `metadata` is empty. Using an empty json object `{}` as the field value should resolve this issue. Also, the `id` field will autoincrement, so you may also try leaving the value of `id` as empty.

CLIENT_FETCH_ERROR

If you experience this error, it may be the way the default Auth callback in the server is using the WEBAPP_URL as a base url. The container does not necessarily have access to the same DNS as your local machine, and therefore needs to be configured to resolve to itself. You may be able to correct this by configuring `NEXTAUTH_URL=http://localhost:3000/api/auth`, to help the backend loop back to itself.

```
docker-calcom-1  | @calcom/web:start: [next-auth][error][CLIENT_FETCH_ERROR]
docker-calcom-1  | @calcom/web:start: https://next-auth.js.org/errors#client_fetch_error
request to http://testing.localhost:3000/api/auth/session failed, reason: getaddrinfo
ENOTFOUND testing.localhost {
docker-calcom-1  | @calcom/web:start:   error: {
docker-calcom-1  | @calcom/web:start:     message: 'request to
http://testing.localhost:3000/api/auth/session failed, reason: getaddrinfo ENOTFOUND
testing.localhost',
docker-calcom-1  | @calcom/web:start:     stack: 'FetchError: request to
http://testing.localhost:3000/api/auth/session failed, reason: getaddrinfo ENOTFOUND
testing.localhost\n' +
docker-calcom-1  | @calcom/web:start:       '    at ClientRequest.<anonymous>
(/calcom/node_modules/next/dist/compiled/node-fetch/index.js:1:65756)\n' +
docker-calcom-1  | @calcom/web:start:       '    at ClientRequest.emit (node:events:513:28)\n'
```

```
+
docker-calcom-1  | @calcom/web:start:       '    at ClientRequest.emit (node:domain:489:12)\n'
+
docker-calcom-1  | @calcom/web:start:       '    at Socket.socketErrorListener
(node:_http_client:494:9)\n' +
docker-calcom-1  | @calcom/web:start:       '    at Socket.emit (node:events:513:28)\n' +
docker-calcom-1  | @calcom/web:start:       '    at Socket.emit (node:domain:489:12)\n' +
docker-calcom-1  | @calcom/web:start:       '    at emitErrorNT
(node:internal/streams/destroy:157:8)\n' +
docker-calcom-1  | @calcom/web:start:       '    at emitErrorCloseNT
(node:internal/streams/destroy:122:3)\n' +
docker-calcom-1  | @calcom/web:start:       '    at processTicksAndRejections
(node:internal/process/task_queues:83:21)',
docker-calcom-1  | @calcom/web:start:     name: 'FetchError'
docker-calcom-1  | @calcom/web:start:   },
docker-calcom-1  | @calcom/web:start:   url: 'http://testing.localhost:3000/api/auth/session',
docker-calcom-1  | @calcom/web:start:   message: 'request to
http://testing.localhost:3000/api/auth/session failed, reason: getaddrinfo ENOTFOUND
testing.localhost'
docker-calcom-1  | @calcom/web:start: }
```

Railway

Deploy on Railway

You can deploy Cal.diy on Railway. The team at Railway also have a detailed blog post on deploying on their platform.

Northflank

Deploy on Northflank

You can deploy Cal.diy on Northflank. The team at Northflank also have a detailed blog post on deploying on their platform.

Vercel

Currently Vercel Pro Plan is required to be able to Deploy this application with Vercel, due to limitations on the number of serverless functions on the free plan.

Deploy with Vercel

Render

Deploy to Render

Elestio

Deploy on Elestio

License

Cal.diy is fully open source, licensed under the MIT License.

Unlike Cal.com's "Open Core" model, Cal.diy has **no commercial/enterprise code**. The entire codebase is available under the same open-source license.

Enabling Content Security Policy

- Set CSP_POLICY="non-strict" env variable, which enables Strict CSP except for `unsafe-inline` in `style-src`. If you have custom changes in your instance, you may need to modify your code to make it CSP-compatible. Currently, strict CSP is enabled only on the login page. On other SSR pages, it is enabled in report-only mode to detect potential issues. It is not yet supported on SSG pages.

Integrations

Obtaining the Google API Credentials

1. Open Google API Console. If you don't have a project in your Google Cloud subscription, you'll need to create one before proceeding further. Under Dashboard pane, select Enable APIS and Services.
2. In the search box, type calendar and select the Google Calendar API search result.
3. Enable the selected API.
4. Next, go to the OAuth consent screen from the side pane. Select the app type (Internal or External) and enter the basic app details on the first page.
5. In the second page on Scopes, select Add or Remove Scopes. Search for Calendar.event and select the scope with scope value `.../auth/calendar.events`, `.../auth/calendar.readonly` and select Update.
6. In the third page (Test Users), add the Google account(s) you'll be using. Make sure the details are correct on the last page of the wizard and your consent screen will be configured.
7. Now select Credentials from the side pane and then select Create Credentials. Select the OAuth Client ID option.
8. Select Web Application as the Application Type.
9. Under Authorized redirect URI's, select Add URI and then add the URI `<Cal.diy URL>/api/integrations/googlecalendar/callback` and `<Cal.diy URL>/api/auth/callback/google` replacing Cal.diy URL with the URI at which your application runs.
10. The key will be created and you will be redirected back to the Credentials page. Select the newly generated client ID under OAuth 2.0 Client IDs.
11. Select Download JSON. Copy the contents of this file and paste the entire JSON string in the `.env` file as the value for `GOOGLE_API_CREDENTIALS` key.

Adding google calendar to Cal.diy App Store

After adding Google credentials, you can now add the Google Calendar app to the App Store. You can repopulate the App Store by running

```
cd packages/prisma
yarn seed-app-store
```

You will need to complete a few more steps to activate Google Calendar App. Make sure to complete section "Obtaining the Google API Credentials". After that do the following

1. Add extra redirect URL `<Cal.diy URL>/api/auth/callback/google`
2. Under 'OAuth consent screen', click "PUBLISH APP"

Obtaining Microsoft Graph Client ID and Secret

1. Open Azure App Registration and select New registration
2. Name your application
3. Set **Who can use this application or access this API?** to **Accounts in any organizational directory (Any Azure AD directory - Multitenant)**
4. Set the **Web** redirect URI to `<Cal.diy URL>/api/integrations/office365calendar/callback` replacing Cal.diy URL with the URI at which your application runs.
5. Use **Application (client) ID** as the **MS_GRAPH_CLIENT_ID** attribute value in .env
6. Click **Certificates & secrets** create a new client secret and use the value as the **MS_GRAPH_CLIENT_SECRET** attribute

Obtaining Zoom Client ID and Secret

1. Open Zoom Marketplace and sign in with your Zoom account.
2. On the upper right, click "Develop" => "Build App".
3. Select "General App" , click "Create".
4. Name your App.
5. Choose "User-managed app" for "Select how the app is managed".
6. De-select the option to publish the app on the Zoom App Marketplace, if asked.
7. Now copy the Client ID and Client Secret to your `.env` file into the `ZOOM_CLIENT_ID` and `ZOOM_CLIENT_SECRET` fields.
8. Set the "OAuth Redirect URL" under "OAuth Information" as `<Cal.diy URL>/api/integrations/zoomvideo/callback` replacing Cal.diy URL with the URI at which your application runs.
9. Also add the redirect URL given above as an allow list URL and enable "Subdomain check". Make sure, it says "saved" below the form.
10. You don't need to provide basic information about your app. Instead click on "Scopes" and then on "+ Add Scopes". On the left,
    1. click the category "Meeting" and check the scope `meeting:write:meeting`.
    2. click the category "User" and check the scope `user:read:settings`.
11. Click "Done".
12. You're good to go. Now you can easily add your Zoom integration in the Cal.diy settings.

Obtaining Daily API Credentials

1. Open Daily.co and create an account.
2. From within your dashboard, go to the developers tab.
3. Copy your API key.
4. Now paste the API key to your `.env` file into the `DAILY_API_KEY` field in your `.env` file.
5. If you have the Daily Scale Plan set the `DAILY_SCALE_PLAN` variable to `true` in order to use features like video recording.

Obtaining Basecamp Client ID and Secret

1. Visit the 37 Signals Integrations Dashboard and sign in.
2. Register a new application by clicking the Register one now link.
3. Fill in your company details.
4. Select Basecamp 4 as the product to integrate with.
5. Set the Redirect URL for OAuth `<Cal.diy URL>/api/integrations/basecamp3/callback` replacing Cal.diy URL with the URI at which your application runs.
6. Click on done and copy the Client ID and secret into the `BASECAMP3_CLIENT_ID` and `BASECAMP3_CLIENT_SECRET` fields.
7. Set the `BASECAMP3_CLIENT_SECRET` env variable to `{your_domain} ({support_email})`.

Obtaining HubSpot Client ID and Secret

1. Open HubSpot Developer and sign into your account, or create a new one.
2. From within the home of the Developer account page, go to "Manage apps".
3. Click "Create legacy app" button top right and select public app.
4. Fill in any information you want in the "App info" tab
5. Go to tab "Auth"
6. Now copy the Client ID and Client Secret to your `.env` file into the `HUBSPOT_CLIENT_ID` and `HUBSPOT_CLIENT_SECRET` fields.
7. Set the Redirect URL for OAuth `<Cal.diy URL>/api/integrations/hubspot/callback` replacing Cal.diy URL with the URI at which your application runs.
8. In the "Scopes" section at the bottom of the page, make sure you select "Read" and "Write" for scopes called `crm.objects.contacts` and `crm.lists`.
9. Click the "Save" button at the bottom footer.
10. You're good to go. Now you can see any booking in Cal.diy created as a meeting in HubSpot for your contacts.

Obtaining Webex Client ID and Secret

See Webex Readme

Obtaining ZohoCRM Client ID and Secret

1. Open Zoho API Console and sign into your account, or create a new one.
2. From within the API console page, go to "Applications".
3. Click "ADD CLIENT" button top right and select "Server-based Applications".
4. Fill in any information you want in the "Client Details" tab

5. Go to tab "Client Secret" tab.
6. Now copy the Client ID and Client Secret to your `.env` file into the `ZOHOCRM_CLIENT_ID` and `ZOHOCRM_CLIENT_SECRET` fields.
7. Set the Redirect URL for OAuth `<Cal.diy URL>/api/integrations/zohocrm/callback` replacing Cal.diy URL with the URI at which your application runs.
8. In the "Settings" section check the "Multi-DC" option if you wish to use the same OAuth credentials for all data centers.
9. Click the "Save"/ "UPDATE" button at the bottom footer.
10. You're good to go. Now you can easily add your ZohoCRM integration in the Cal.diy settings.

Obtaining Zoho Calendar Client ID and Secret

Follow these steps

Obtaining Zoho Bigin Client ID and Secret

Follow these steps

Obtaining Pipedrive Client ID and Secret

Follow these steps

Rate Limiting with Unkey

Cal.diy uses Unkey for rate limiting. This is an optional feature and is not required for self-hosting.

If you want to enable rate limiting:

1. Sign up for an account at unkey.com
2. Create a Root key with permissions for `ratelimit.create_namespace` and `ratelimit.limit`
3. Copy the root key to your `.env` file into the `UNKEY_ROOT_KEY` field

Note: If you don't configure Unkey, Cal.diy will work normally without rate limiting enabled.

Contributing

We welcome contributions! Whether it's fixing a typo, improving documentation, or building new features, your help makes Cal.diy better.

> **Important:** Cal.diy is a community fork. Contributions to this repo do **not** flow to Cal.com's production platform. See CONTRIBUTING.md for details.

- Check out our Contributing Guide for detailed steps.
- Please follow our coding standards and commit message conventions to keep the project consistent.

Even small improvements matter — thank you for helping us grow!

Good First Issues

We have a list of help wanted that contain small features and bugs which have a relatively limited scope. This is a great place to get started, gain experience, and get familiar with our contribution process.

Revision #2
Created 2026-08-24 10:25:44 UTC by Administrador
Updated 2026-08-24 11:27:54 UTC by Administrador