Pipelines

- Upload files automatically to SFTP server
- Build and deploy node.js app automatically using CI/CD

Upload files automatically to SFTP server

> Prerequisite: A Gitlab Runner must be available in the project or group. You also need to know the information on how access to your (S)FTP server.

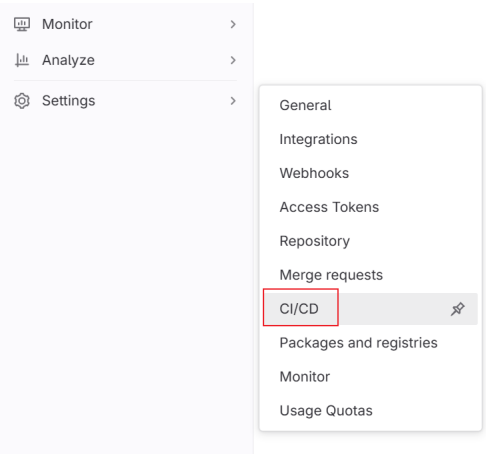
This article describes how to automatically upload files to an (S)FTP server. We will create a pipeline in the Gitlab project that runs on a runner and uploads the files. The intended use could be to publish a static website on a web server or to keep information on the Internet up to date.

We will use an (S)FTP Linux client for the upload, which is executed on the command line.

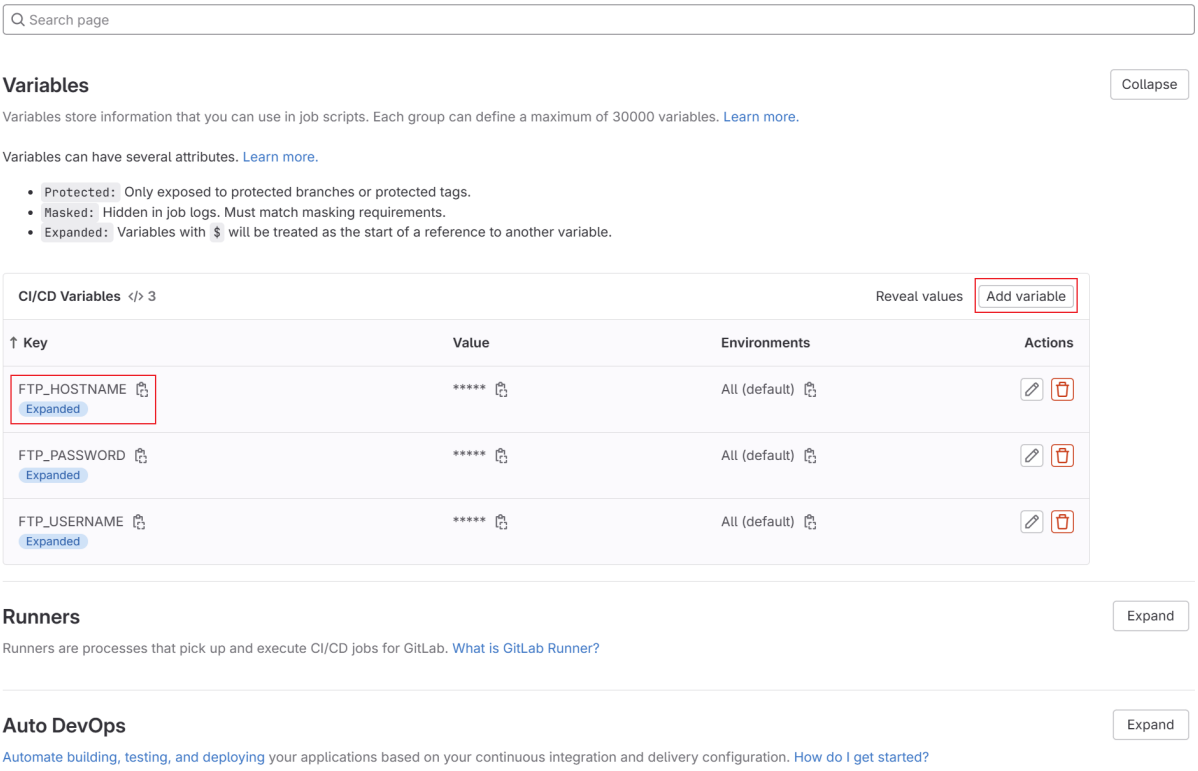
Store credentials as a variable

To ensure that the credentials for your (S)FTP server can be managed centrally and are not stored as plain text in your repo, it is strongly recommended to use the variable function. To do this, variables can be created in the project or at group level, which can then be used in the pipeline using $VARIABLE.

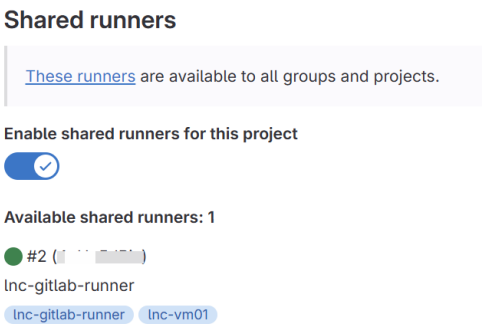
Go to your group or project in your Gitlab instance or in the cloud. You can choose Settings -> CI/CD to open the variable secion.



You can then enter your access data under "Variables". Make sure you use the correct name in the code. For credential data, it is recommended to deactivate the "Protected" setting and to active"Expand".



In the same view, you can check that a Gitlab Runner is available for the project or group. The view should look like this:



Create pipeline

In the project that you want to upload to the server, you must create a .gitlab-ci.yml file. You can then copy the following code into it.

```
image: ubuntu:22.04


stages:
  - before_script
```

```
  - deploy

before_script:
  - apt update -qy
  - apt install -y lftp

deploy:
  stage: deploy
  script:
    - lftp -e "set ssl:verify-certificate no; open $FTP_HOSTNAME; user $FTP_USERNAME $FTP_PASSWORD; mirror -
X .* -X .*/ --reverse --verbose --delete . ./<yourdestinationfolder>; bye"
  only:
    - master
    - main
```

Make sure that you set the target path correctly and you may want to add other parameters so that it works with your FTP setup. If you have created the file correctly, you can commit the code to your GIT repo and push it to the Gitlab remote repository. The code is then executed and your repo files are uploaded to the (S)FTP server.

The Pipeline now only executes this configuration if it is pushed into the Master or Main branch. However, you can change this relatively quickly by removing the code block.

Build and deploy node.js app automatically using CI/CD

> Prerequisites: A Gitlab Runner must be available in the project or group. Your code must be in the repo on Gitlab instance or cloud.

This guide enables automation in the build and deployment process. Tested frameworks with this solution are: SvelteKit, Express.JS, Next.JS.

Use case

If a node.js application has regular feature updates or bug fixes, it can be tedious to deploy them manually.
Here you can find an example pipeline of how building and deploying the website using Docker containers could work.
However, if you only want to upload the files to your web server via (S)FTP, you can combine this guide and the following one: Upload files automatic... | LNC DOCS (lucanoahcaprez.ch)

Architecture

The building and deploying steps should be possible via a pipeline. First, a container is created via the Gitlab Runner, which builds the code. This container is then finalized and uploaded to a registry.

Once the Docker image has arrived in the registry, the building container is destroyed. The next step is then to use docker-compose and the corresponding image that we created previously on the production server.

Pipeline

The following code could be an example pipeline that first builds and containerizes the application and then deploys it using docker-compose. Save the following code as a ".gitlab-ci.yml" document in the main level of your GIT project.

Note that you need to specify the Docker hosts for the build process and for the production release. In addition, you must provide the following variables correctly in the project or in the group.

- $REGISTRY_URL
- $REGISTRY_USERNAME
- $REGISTRY_PASSWORD

More on Gitlabs CI/CD variables can be found here: Upload files automatic... | LNC DOCS (lucanoahcaprez.ch)

```
stages:
  - containerize
  - deploy

containerize:
  variables:
     DOCKER_HOST: tcp://<yourdockerbuildmachine>:2375
  image: docker
  stage: containerize
  services:
    - docker:dind
  script:
    - apk add --update nodejs npm
    - npm i
    - npm run build
    - docker build -t $REGISTRY_URL/<yourprojectname> .
    - echo $REGISTRY_PASSWORD | docker login https://$REGISTRY_URL -u $REGISTRY_USERNAME --password-stdin
    - docker push $REGISTRY_URL/<yourprojectname>
    - ls -la .
  only:
    - master
    - main

deploy:
  variables:
    DOCKER_HOST: tcp://<yourdockerhostingmachine>:2375
```

```
  image: docker
  stage: deploy
  script:
    - docker-compose up -d
  only:
    - master
    - main
```

Deployment configuration

Save these files in addition to the .gitlab-ci.yml configuration at the top level.

docker-compose.yml

When publishing, docker-compose uses a configuration file. This must be located in the top directory of the GIT repo. This configuration can be extended as required and only serves as a basis for publishing.

```
version: '3'
services:
  node:
    image: '<yourregistryurl>/<yourprojectname>:latest'
    container_name: <yourprefereddockername>
    restart: unless-stopped
    ports:
      - '<hostport>:3000'
```

Dockerfile

The last file that is required is the Dockerfile. This file tells the build process what the docker container should ultimately look like. Here is a generic Dockerfile that builds a node app and makes it available on port 3000.

```
FROM node:18-alpine

RUN mkdir -p /usr/app/
WORKDIR /usr/app/

COPY ./ ./
```

```
RUN npm install
RUN npm run build

EXPOSE 3000
CMD ["node", "build/index.js"]
```