Optimize Docker Offload usage

- dockerignore files
- Slim base images
- Multi-stage builds
- Fetch remote files in build
  - Multi-threaded tools

Docker Offload builds and runs your containers remotely, not on the machine where you invoke the commands. This means that files must be transferred from your local system to the cloud over the network.

Transferring files over the network introduces higher latency and lower bandwidth compared to local transfers.

Even with optimizations, large projects or slower network connections can lead to longer transfer times. Here are several ways to optimize your setup for Docker Offload:

- Use `.dockerignore` files
- Choose slim base images
- Use multi-stage builds
- Fetch remote files during the build
- Leverage multi-threaded tools

For general Dockerfile tips, see Building best practices.

dockerignore files

A `.dockerignore` file lets you specify which local files should *not* be included in the build context. Files excluded by these patterns won't be uploaded to Docker Offload during a build.

Typical items to ignore:

- `.git` – avoids transferring your version history. (Note: you won't be able to run `git` commands in the build.)
- Build artifacts or locally generated binaries.
- Dependency folders such as `node_modules`, if those are restored in the build process.

As a rule of thumb, your `.dockerignore` should be similar to your `.gitignore`.

Slim base images

Smaller base images in your `FROM` instructions can reduce final image size and improve build performance. The `alpine` image is a good example of a minimal base.

For fully static binaries, you can use `scratch`, which is an empty base image.

Multi-stage builds

Multi-stage builds let you separate build-time and runtime environments in your Dockerfile. This not only reduces the size of the final image but also allows for parallel stage execution during the build.

Use `COPY --from` to copy files from earlier stages or external images. This approach helps minimize unnecessary layers and reduce final image size.

Fetch remote files in build

When possible, download large files from the internet during the build itself instead of bundling them in your local context. This avoids network transfer from your client to Docker Offload.

You can do this using:

- The Dockerfile `ADD` instruction
- `RUN` commands like `wget`, `curl`, or `rsync`

Multi-threaded tools

Some build tools, such as `make`, are single-threaded by default. If the tool supports it, configure it to run in parallel. For example, use `make --jobs=4` to run four jobs simultaneously.

Taking advantage of available CPU resources in the cloud can significantly improve build time.

□□□ #2
□ □□□ □□□ 2026-10-31 19:35:24 UTC
□ □□□ □□□ 2026-10-31 19:35:25 UTC