Docker 上で Codex・Claude Code・Gemini を統一操作する agent コマンド

著者：関 勝寿　公開日：2026 年 9 月 20 日　ソース：https://sekika.github.io/2026/09/20/agent/

agent は、Docker 上で Codex、Claude Code、Gemini / Antigravity のいずれかを実行するための単一コマンドです。エージェントごとに異なる起動手順や、コンテナの作成・再利用・削除を覚えなくても、同じ操作体系で開発用エージェントを使えるようにすることが目的です。

コンテナを使うことで、ホスト環境を汚さずに実行環境を分離できます。作業ディレクトリだけをコンテナへ共有し、エージェントごとの設定・認証状態はホスト側に保持します。プロジェクト固有のパスや、特定ホストだけの設定には依存しません。

1. インストールとセットアップ

1.1. 前提条件

- **Docker**: Docker がインストールされており、Docker デーモンが動作していること（現在のユーザーで docker コマンドが実行できる権限があること）。
- **Python 3**: ホスト環境に Python 3（Python 3.8 以上推奨）がインストールされていること。

1.2.1. スクリプトのダウンロード

curl または wget を使用して、agent スクリプトをダウンロードします。

```
# curl を使用する場合
curl -fsSL https://sekika.github.io/file/agent/agent -o agent

# または wget を使用する場合
wget https://sekika.github.io/file/agent/agent
```

1.3.2. 実行権限の付与

ダウンロードした agent ファイルに実行権限を付与します。

```
chmod +x agent
```

1.4.3. PATH の通ったディレクトリへの配置

ターミナルからいつでも agent コマンドとして呼び出せるよう、PATH の通ったディレクトリに配置します。たとえば

```
sudo mv agent /usr/local/bin/
```

1.5.4. Dockerfile のダウンロード

agent は各エージェントを実行する Docker コンテナ環境を構築するために Dockerfile を使用します。プロジェクトディレクトリや作業ディレクトリなど、任意の場所に Dockerfile をダウンロードします。

```
# curl を使用する場合
curl -fsSL https://sekika.github.io/file/agent/Dockerfile -o Dockerfile

# または wget を使用する場合
wget https://sekika.github.io/file/agent/Dockerfile
```

配布する Dockerfile は、既定では Debian Trixie の slim イメージを使用します。

```
# https://hub.docker.com/_/debian
FROM debian:trixie-slim
```

Debian の別のリリースを使用する場合は、FROM 行のイメージタグを使用したいリリースに合わせて適宜書き換えてください。利用可能なタグは Docker Hub の Debian 公式イメージで確認できます。

1.6.5. コンテナイメージのビルド

ダウンロードした Dockerfile をもとに、agent build コマンドで Docker イメージ（既定値：agent:1.0）をビルドします。

```
# カレントディレクトリに Dockerfile がある場合
agent build

# Dockerfile を別の場所に配置している場合
agent build --dockerfile /path/to/Dockerfile
```

通常の build では Docker のキャッシュが利用されます。OS のベースイメージから完全に作り直したい場合は、agent rebuild を使用します。rebuild では --no-cache --pull を付けて、キャッシュを破棄して最新のベースイメージから再構築します。

```
agent rebuild
```

既存イメージをベースとしてパッケージと各 CLI だけを最新化したい場合は、agent update を使用します。パッケージに対して apt-get upgrade を実行し、Codex、Claude Code、Gemini CLI を @latest へ更新するとともに、Antigravity CLI も更新したイメージを作成します。更新用の Dockerfile は別ファイルとして作成せず、標準入力から Docker に渡します。

```
agent update
```

対象となるイメージがまだ存在しない場合、agent update は通常の build と同じ方法で Dockerfile からイメージを作成します。

1.7.6. 動作確認

インストールおよびイメージの作成が正常に完了したか確認します。

```
# ヘルプの表示確認
agent --help

# 作成されたイメージの確認
agent images
```

ヘルプやイメージ一覧が表示されればセットアップは完了です。

2. 基本的な使い方

引数なし、または run を指定すると、選択したエージェントをコンテナ内で実行します。

```
# Codex（既定値）
agent
agent --agent codex

# Claude Code / Gemini
agent --agent claude
agent --agent gemini

# 短縮名も使用可能
```

```
agent --agent cx
agent --agent cl
agent --agent gm

# モデル、推論強度、作業ディレクトリを指定
agent -m gpt-5 --effort high
agent --dir /path/to/project

# 起動前にイメージ内のパッケージと CLI を更新
agent -u
```

agent -u は、エージェントを起動する前に agent update と同じ方法で既存イメージを更新し、更新したイメージからコンテナを作り直してエージェントを起動します。

指定した作業ディレクトリは、コンテナ内の --workspace で指定した場所へマウントされます。選択したエージェントに応じて、必要な設定・認証状態もコンテナへマウントされます。

--dir（または -d）で作業ディレクトリを指定しない場合は、agent コマンドを実行した時点のホスト側のカレントディレクトリが作業ディレクトリになります。そのディレクトリがコンテナ内の --workspace（既定値は /workspace）へマウントされます。

gemini（短縮名 gm）を選択した場合、コンテナ内では Antigravity CLI の agy をエントリポイントとして使用します。

3. サブコマンド

- run（既定）：コンテナを準備し、選択した CLI を実行します。後ろの引数は CLI に渡されます。
- build: --dockerfile で指定した Dockerfile から、--image のタグでイメージを作成します。Docker のキャッシュを利用し、ビルドコンテキストは --dir です。
- rebuild: Dockerfile から --no-cache --pull を付けてビルドし、キャッシュを破棄してベースイメージから完全に再作成します。
- update: エージェントを起動せず、既存イメージをベースとしてパッケージと各 CLI を更新したイメージを作成します。対象イメージが存在しない場合は通常の build を行います。
- start: エージェントを実行せず、コンテナを作成または起動します。
- stop: 指定コンテナを停止します。
- rm: 指定コンテナを強制削除します。イメージとホスト側設定は削除しません。
- shell: 指定ユーザーでコンテナ内の Bash を対話的に起動します。
- exec: コンテナ内で任意のコマンドを実行します。
- images: docker images で Docker イメージを一覧表示します。

```
agent build
agent rebuild
agent update
agent start
agent shell
agent exec -- ls -la
agent stop
agent rm
agent images
```

exec の -- は agent 自身の引数と、コンテナ内コマンドの引数を区切ります。コマンドを省略するとエラーになります。

build、rebuild、update の違いをまとめると、build は Dockerfile からの通常のビルド、rebuild はキャッシュを破棄して最新のベースイメージから行う完全な再ビルド、update は既存イメージをベースとしてパッケージと CLI を更新するためのビルドです。

4. オプション

オプション	内容
-a, --agent	codex (cx) / claude (cl) / gemini (gm)。
-d, --dir	作業ディレクトリ。実行時のマウント元とビルドコンテキスト。省略時は agent を実行した時点のホスト側のカレントディレクトリ。
--dockerfile	build / rebuild で使う Dockerfile。既定値は ./Dockerfile。
--image	使用・作成するイメージ名。既定値は agent:1.0。
--container	コンテナ名。既定値は agent。
--user	コンテナ内の実行ユーザー。既定値は agent。
--workspace	コンテナ内の作業ディレクトリ。既定値は /workspace。
-m, --model	選択したエージェントに渡すモデル名。
--effort	推論強度。Codex / Claude は low / medium / high / xhigh、Gemini / Antigravity は low / medium / high。
-u, --update	起動前に既存イメージをベースとしてパッケージと CLI を更新し、更新したイメージからコンテナを作り直します。
--dry-run	Docker を実行せず、指定された agent コマンドラインを表示して終了します。

コマンドラインの値は設定ファイルと組み込み規定値より優先されます。

5. 設定ファイルと自動保存

設定ファイルは ~/.config/agent/agent.ini です。存在しない場合、初回実行時に [agent] の規定値を自動保存します。[models] と [efforts] は必要な場合に追加する任意のセクションです。ホームディレクトリに書き込めない場合は警告を表示し、規定値で処理を続けます。

```
[agent]
agent = codex
dockerfile = ./Dockerfile
image = agent:1.0
container = agent
user = agent
workspace = /workspace
model =
effort =

[models]
codex = gpt-6-astra
claude = claude-opus-5
gemini = gemini-3.8-flash

[efforts]
codex = xhigh
claude = high
gemini = high
```

コマンドラインで一時的に指定した値は、設定ファイルへ自動的には書き戻しません。継続して使う値を変更する場合は、このファイルを編集してください。

model と effort は全エージェント共通のフォールバックです。エー

ジェントごとに異なる値を使う場合は、[models] と [efforts] に codex、claude、gemini のキーを指定します。短縮名の cx、cl、gm もキーとして使用できます。

モデルは、コマンドラインの -m / --model、[models] のエージェント別設定、[agent] の model の順で優先されます。推論強度も同様に、コマンドラインの --effort、[efforts] のエージェント別設定、[agent] の effort の順で優先されます。

6. 設定・認証状態

エージェントごとに次のホスト側ディレクトリを、対応する CLI の設定ディレクトリとしてコンテナへマウントします。

```
Codex:
~/.codex        -> /home/agent/.codex

Claude Code:
~/.claude       -> /home/agent/.claude

Gemini / Antigravity:
~/.gemini       -> /home/agent/.gemini
~/.local/share/keyrings -> /home/agent/.local/share/keyrings
```

認証情報や CLI の設定形式は各 CLI の通常の仕組みに従います。agent は認証情報を変換せず、プロジェクト固有の絶対パスやホスト専用設定を追加しません。

Codex と Gemini / Antigravity については、必要なホスト側ディレクトリが存在しない場合に自動的に作成します。Claude Code の ~/.claude が存在しない場合はエラーとして終了します。

Gemini / Antigravity のコンテナでは、認証処理のためホストネットワークを使用し、keyring もホスト側に保持します。

7.Docker の前提と既定値

Docker コマンドがインストールされ、Docker デーモンへ接続できる必要があります。利用できない場合は、明確なエラーを表示して終了します。

・Dockerfile: ./Dockerfile
・作業ディレクトリ：実行時のカレントディレクトリ
・イメージ：agent:1.0
・コンテナ：agent
・コンテナ内ユーザー：agent
・コンテナ内ワークスペース：/workspace

指定したイメージが存在しない状態で run、start、shell、exec を実行した場合は、コンテナを作成する前に Dockerfile からイメージを自動的にビルドします。

8. 起動時の共通動作

agent は、コンテナを必要とする操作の開始時に既存コンテナを削除して、指定した作業ディレクトリと認証状態を bind mount したコンテナを作り直します。作成済みコンテナの mount は後から変更できないためです。イメージを更新した場合も、既存コンテナには更新内容が反映されないため、更新したイメージからコンテナを作り直します。

作業ディレクトリは既定で起動時のカレントディレクトリです。-d を指定すると、そのディレクトリをコンテナ内の --workspace で指定した場所（既定値 /workspace）に割り当てます。build と rebuild のビルドコンテキストもこのディレクトリです。

Codex のホスト側設定とログイン状態は ~/.codex、Claude Code は ~/.claude、Gemini / Antigravity は ~/.gemini と keyring をコンテナへマウントします。コンテナを作り直しても、これらの状態は保持されます。

コンテナだけを準備する場合は start、停止する場合は stop、削除する場合は rm を使用します。コンテナ内で対話的に Bash を使う場合は shell、任意のコマンドを実行する場合は exec を使用します。

9.Codex のコンテナ起動

コンテナ自体を実行境界として使うため、Codex の追加 sandbox は起動時に sandbox_mode=danger-full-access へ上書きします。コンテナ内ではユーザー名前空間を作成できないため、Codex が使う bubblewrap は実行しません。

ホスト側設定をそのままコンテナへ持ち込むと、コンテナ内で利用できない MCP やプラグインが起動することがあります。そのため、node_repl などコンテナ環境で使用しない機能を起動時のオプションで無効化します。また、Codex の通知機能も起動時に無効化します。これらは起動時の上書きであり、ホスト側の ~/.codex/config.toml は変更しません。