Fabric & Invoke Quick Guide

Invoke : Pythonで記述するタスクランナー

Fabric : SSHを使うデプロイタスクランナー

[1] install

```
PY2  $ pip  install invoke fabric
PY3  $ pip3 install invoke fabric
```

[2] Invokeのタスクの定義

```
from invoke import *

@task
def install(c):
  c.run("pip install -r requirements.txt")
```

ファイル名は "tasks.py"
タスク関数に @task を付ける

[3] タスクの実行

```
$ invoke -e install
$ invoke -e build_lint_package
$ invoke -l
```

tasks.py のあるディレクトリで実行
タスク名をスペースで区切る

"-e" で実行するコマンドを表示
"-l" でタスクの一覧を表示

[4] 繰り返し操作

```
@task
def build(c):
  packages = ["Client", "Server", "Tester"]

  for name in packages:
    c.run(f"go build cmd/{name} -o out/{name}")
```

for文で書くだけ！

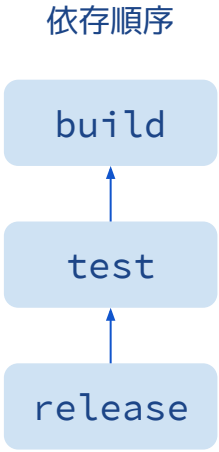
[5] タスクの依存関係

```
@task
def build(c):
  c.run("go build -o out")

@task(build)
def test(c):
  c.run("go test")

@task(test)
def release(c):
  c.run("zip out.zip out/")
```

依存している関数を
@taskの引数に指定する




```
$ invoke -e release
go build -o out
go test
zip out.zip out/
```

最終成果物のタスクを指定
依存順序どおりに実行

[6] 終了コードでの分岐

```
@task
def build(c):
  if c.run("test -e ./node_modules", warn=True)
                                          .failed:
    c.run("npm install")

  if c.run("test -e out", warn=True).ok:
    c.run("rm -rf out")

  c.run("tsc")
```

通常は異常終了すると停止
warn=Trueを付けると処理続行
bool型ok、failedプロパティを元に
if文で記述できる

[7] ディレクトリ移動

```
@task
def build(c):
  with c.cd("client"):
    c.run("go build")
    c.run("go test")

  with c.cd("server"):
    c.run("go build")
    c.run("go test")
```

with句のコンテキストのみ
ディレクトリを移動する

```
$ invoke -e build
cd client && go build
cd client && go test
cd server && go build
cd server && go test
```

[8] パラメータを与える

```
@task
def release(c, gs_path, force=False):
  if force:
    c.run("git push -f")
  else:
    c.run("git push")

  r = c.run(f"gsutil rsync . {gs_path}")
```

```
$ invoke -e release --gs-path gs://release-gcs/
git push
gsutil rsync . gs://release-gcs/

$ invoke -e release --gs-path gs://release-gcs/
                                              --force
git push -f
gsutil rsync . gs://release-gcs/
```

関数の引数がそのままタスク実行時の引数に
初期値も設定できる
なお "_" が "-" に置き換えられる

[9] sedの代わりに使う

戻り値に、stdout、stderr があり
Pythonの文字列として処理できる

```
@task
def release(c):
  r = c.run("cat deployment.yaml")
  yaml = r.stdout.replace("__IP__", "3.4.5.6")

  # まだ標準入力をそのまま渡せないのでファイルに出力
  with os.open("_deployment.yaml") as f:
    f.write(yaml)

  c.run("kubectl apply -f _deployment.yaml")
```

[10] jqの代わりに使う

Pythonのライブラリを使えば
jsonは辞書型で扱える

```
import json

@task
def release(c):
  r = c.run("gcloud output=json compute instances
                                                list")
  l = json.loads(r.stdout)
  for ins in l:
    name = ins["name"]
    c.run("gcloud compute instances delete
                                              {name}")
```

一方、Shell Scriptなら…

```
#!/bin/bash
gcloud --format=json compute instances list |
                  jq .[].name -r | while read name
do
  gcloud compute instances delete $name
done
```

[11] サーバに接続する Fabric のタスク

```
from fabric import *

@task
def release(c):
  c.put("dist.zip", "app/")

  with c.cd("dist"):
    c.run("unzip dist.zip")

  c.sudo("systemctl restart serverapp")
```

ファイル名は "fabfile.py"
タスク関数に @task を付ける
だいたい Invoke と同じ
put、get、sudoなどが使える

```
$ fab -e release -H ec2-users@xxx.amazonaws.com
```

"-H" でホスト名を指定する

[12] Invoke タスクの中で Fabric でデプロイする

```
from invoke import task
from fabric import Connection

@task
def build(c):
  c.run("go build -o out/server")

@task(build)
def release(c):
  conn = Connection("release -H ec2@xxxxxxx")
  conn.put("out/server", "app/")
  conn.sudo("systemctl restart serverapp")
```

fabric.Connection を使う

```
$ invoke -e release
```

[13] 設定ファイルを使う

開発環境用 dev.yaml

```
project: "nnyn-dev"
zone: "asia-northeast1-b"
```

本番環境用 prod.yaml

```
project: "nnyn-prod"
zone: "nnyn-prod-bucket-a"
```

invoke.Context.config で設定にアクセスする

```
from invoke import task

@task
def delete_instance(c):
  l = c.run(f"gcloud --project {c.config.project}
            compute instances list
            --zones={c.config.zone}").stdout
  for i in l.split("\n")
    c.run(f"gcloud compute insances delete {i}")
```

```
$ invoke -e -f dev.yaml delete-instance
gcloud --project nnyn-dev compute instances list
            --zones=asia-northeast1-b

$ invoke -e -f prod.yaml delete-instance
gcloud --project nnyn-prod compute instances list
            --zones=asia-northeast1-a
```

[14] 一次情報

Invoke main http://www.pyinvoke.org/
Invoke doc http://docs.pyinvoke.org/en/latest/
Fabric main http://www.fabfile.org/
Fabric doc http://docs.fabfile.org/en/latest/

[15] 使い分け

Maven、Gradle、npm、Gulp、msbuild...

その言語、プラットフォームに依存したビルドのタスクランナー

Make

汎用ビルドタスクランナー
- 依存関係が記述できる
- ファイル単位の成果物管理のため、合わないケースもある

ShellScript

Unix環境で必ず動作できる汎用スクリプト
- 環境構築が不要
- ちょっとした文字列操作にもsedなどUnixツールが必要

Windowsでは基本動かない（Cygwin、WSLが必要）

Invoke

Pythonで動く汎用タスクランナー

Python言語のため文字列操作等が豊富
- パッケージのインストールが必要
- 依存関係が記述できる

MacOS、Linux、Windowsで同じように扱える
- プログラミング言語のため、好きなレベルの抽象化ができる
- 単純なため習得が楽
- 小さい用途でも使い始められる

Ansible, Chef...

サーバの構成管理ツール
- サーバの状態の定義を記述する
- ちょっとしたリリース用途にはオーバースペックなことも

Fabric

SSHを用いて、リモートサーバへのデプロイ及び、
- タスクを実行したい場合に使う
- 特定のポートを繋げてコマンドを実行することもできる

Fabric3

Fabric v1のPython3対応folkの名前
- 今のv2からPython3対応済みなので無視すること

Visual Studio Code デバッグ技術

技術書典3で頒布した本を
2018年秋現在の状況に改定して出版！
Ruby、Reactなど多くを追加！

Golang、AppEngine(Go)、Python、C#、
NodeJS、TypeScript、Chrome、Electron、React、
Ruby、C/C++、Java、PHP、Bash

出版社：インプレスR&D
出版形式：Amazonプリントオンデマンド、Kindle他

Amazon他で
近日発刊！

ShellScriptの代わりに Fabric & Invoke Pythonタスクランナーを 活用する技術

技術書典5にて頒布
80Pまるごと全部Fabric&Invoke
だいたいこのPaperで紹介したけど、他に、
入力が要るコマンド/NameSpace/ポート接続……

構造化と性能の間をGolangで攻める技術 （+WebWorker活用技術）

技術書典4にて頒布
Golangを選ぶからには
信頼できる性能がほしい
Golangで、性能を取りつつ構造化を図ると
どの程度のコストがかかるのかを、
実測して見極めて考える本
+WebWorkerでのタスク並列化の有効性の検証記事

このPaperを書いた人

74th （ななよん）
twitter、github : @74th

下2つは
Boothにて販売中！
https://74th.booth.pm/