📘 Python Structure Guide for Modal Cloud (Updated + Endpoints Explanation)

This guide summarizes the complete structure of a Python app on **Modal**, explaining how decorators like `@app.function()`, `@app.local_entrypoint()`, and `@modal.fastapi_endpoint()` work together to define cloud, local, and web execution flows.

🧰STRUCTURE GLOBALE D'UNE APP PYTHON SUR MODAL

```python
import modal

# 1️⃣ Déclaration de l'application
app = modal.App("my_modal_app")

# 2️⃣ (Optionnel) Définition d'une image personnalisée (⚠️ pas un décorateur !)
image = (
    modal.Image.debian_slim()
    .apt_install("ffmpeg")
    .pip_install("requests", "pillow")
)

# 3️⃣ Fonctions cloud (exécutées sur les serveurs Modal)
@app.function(image=image)
def my_function(param: str):
    import requests
    response = requests.get("https://example.com")
    print("Processing:", param)
    return f"Done with {param}"

# 4️⃣ Point d'entrée local (le "main" de ton app)
@app.local_entrypoint()
def main():
    print("🚀 Starting main entrypoint...")
    result = my_function.remote("Task A")
    print("✅ Result:", result)
```

🧱DIFFERENCE ENTRE IMAGE ET DÉCORATEUR

Élément	Type	Rôle	Exemple
`image = modal.Image.debian_slim().pip_install(...)`	Instanciation d'objet	Crée ton environnement Docker personnalisé	`image = Image.debian_slim().p`
`@app.function(image=image)`	Décorateur	Lie la fonction à l'image définie plus haut et l'enregistre dans Modal	`@app.function(image=`

⚠️**En clair :** - `image = ...` → construit un **objet Python** représentant ton runtime cloud (comme un `Dockerfile` auto-généré). - `@app.function(image=image)` → **décorateur** qui enregistre la fonction pour s'exécuter **dans cette image**.

Sous le capot, Modal traduit ton image en quelque chose comme :

```
FROM debian:bookworm-slim
RUN apt-get update && apt-get install -y ffmpeg
RUN pip install requests pillow
```

🗔**Schéma visuel rapide :**

```
┌──────────────────────────────────┐
│  image = Image.debian_slim()     │   ← création d'une image Docker custom
│          .pip_install(...)       │
└──────────────┬───────────────────┘
               │
               ▼
@app.function(image=image)      ← décorateur : attache l'image à la fonction
def my_function(...):           ← fonction cloud
    ...
```

🩲LES ENDPOINTS HTTP DANS MODAL (GET / POST)

Depuis 2025, Modal permet de transformer directement tes fonctions Python en **API FastAPI publiques** via `@modal.fastapi_endpoint()`.

⚙️Exemple complet

```
import modal
from modal import App, Image

app = App(image=Image.debian_slim().pip_install("fastapi[standard]"))

@app.function()
@modal.fastapi_endpoint(docs=True)
def greet(user: str):
    return f"Hello {user}!"

@app.function()
@modal.fastapi_endpoint(method="POST", docs=True)
def square(item: dict):
    return {"value": item['x']**2}
```

🔍COMPORTEMENT DES MÉTHODES HTTP

Modal reprend la logique de **FastAPI** :

Méthode HTTP	Quand l'utiliser	Où vont les données
GET	Lire une info (lecture, test, message)	Les données sont dans l'URL (`?param=...`)
POST	Envoyer des données au serveur (création, calcul, ML inference...)	Les données sont dans le corps JSON (`-d '{...}'`)

🧰Exemple GET

```
@app.function()
@modal.fastapi_endpoint(docs=True)
def greet(user: str):
    return f"Hello {user}!"
```

🕰️Appel client :

```
curl "https://wearekhepri--endpoint-py-greet-dev.modal.run?user=Barbara"
```

➡️Résultat : `{ "result": "Hello Barbara!" }`

🀄Ici, FastAPI extrait automatiquement `user` depuis l'URL (`?user=`) et appelle ta fonction avec ce paramètre.

🧰Exemple POST

```
@app.function()
@modal.fastapi_endpoint(method="POST", docs=True)
def square(item: dict):
    return {"value": item["x"] ** 2}
```

🕰️Appel client :

```
curl -X POST "https://wearekhepri--endpoint-py-square-dev.modal.run"
     -H "Content-Type: application/json"
     -d '{"x": 5}'
```

➡️Résultat : `{ "value": 25 }`

🈸Ici, FastAPI lit le **corps JSON** et le mappe automatiquement sur l'argument `item: dict`.

COMMENT ÇA MARCHE EN INTERNE

1️⃣ **Modal construit ton image Docker** (avec FastAPI inclus) 2️⃣ **Un mini serveur FastAPI** est lancé automatiquement dans le conteneur 3️⃣ **Chaque fonction décorée** devient une route (`/greet`, `/square`, etc.) 4️⃣ **Modal gère l'hébergement, les logs et la scalabilité**

Flux général :

```
Client HTTP (curl / app / navigateur)
        │
        ▼
→ URL publique Modal (FastAPI auto)
        │
        ▼
→ Exécution de ta fonction Python dans le conteneur cloud
        │
        ▼
→ Réponse JSON renvoyée au client
```

🗒️PARAMÈTRES ET TYPES DE DONNÉES

Modal lit automatiquement tes **type hints Python** pour déterminer comment parser la requête :

Signature	Source de données	Exemple
`def greet(user: str)`	Query string (`GET`)	`/greet?user=Barbara`
`def square(item: dict)`	Body JSON (`POST`)	`{"x": 5}`
`def calc(a: int, b: int)`	Query string	`/calc?a=2&b=3`
`def predict(data: dict)`	Body JSON	`-d '{"data": {...}}'`

⬜ FastAPI + Modal gèrent automatiquement la validation et les erreurs de type.

⚙️DOCS AUTOMATIQUES

Le paramètre `docs=True` dans :

```
@modal.fastapi_endpoint(docs=True)
```

active automatiquement **Swagger UI**, consultable à :

```
https://<endpoint>.modal.run/docs
```

Cela te permet de tester tes routes directement dans ton navigateur.

🧹DIFFERENCE ENTRE `modal run` ET `modal serve`

Commande	Rôle	Quand l'utiliser
`modal run script.py`	Exécute une app ponctuelle via `@app.local_entrypoint()`	Tâches unitaires, traitement batch, jobs ML
`modal serve script.py`	Lance un serveur FastAPI et expose les endpoints HTTP	Pour créer une API ou un backend persistant

⬜ `modal serve` surveille ton dossier local : toute modification de ton script redéploie automatiquement le serveur cloud.

🧹STRUCTURE DES URLS MODAL

Lors du `modal serve`, chaque fonction exposée via `@modal.fastapi_endpoint()` reçoit automatiquement une **URL publique** de la forme :

```
https://<org>--<fichier>--<fonction>-<mode>.modal.run
```

Exemple concret

```
https://wearekhepri--endpoint-py-square-dev.modal.run
```

Partie	Signification
`wearekhepri`	Nom de ton organisation / workspace Modal
`endpoint-py`	Nom du fichier Python (`endpoint.py`)
`square`	Nom de la fonction exposée
`dev`	Environnement de développement (`modal serve`)
`.modal.run`	Domaine public géré par Modal

Chaque fonction décorée devient une route indépendante. Exemple :

```
https://wearekhepri--endpoint-py-greet-dev.modal.run
https://wearekhepri--endpoint-py-square-dev.modal.run
```

⬜ En production (`modal deploy`), le suffixe `-dev` disparaît et l'URL devient permanente.

🧰AJOUTER UNE ROUTE RACINE `/`

Par défaut, FastAPI (et donc Modal) **n'a pas de route** `/`. Si tu veux qu'un message s'affiche sur l'URL racine, définis-le manuellement :

```
@app.function()
@modal.fastapi_endpoint(route="/", docs=True)
def home():
    return {"message": "API is running on Modal ✅"}
```

Tu pourras alors aller sur :

```
https://wearekhepri--endpoint-py-home-dev.modal.run
```

ou directement sur :

```
https://wearekhepri--endpoint-py-home-dev.modal.run/
```

➡️et voir : `{ "message": "API is running on Modal ✅" }`

🔗 EN RÉSUMÉ

Élément	Rôle
`App()`	Crée une app Modal
`Image()`	Définit ton environnement d'exécution cloud
`@app.function()`	Code exécuté sur le cloud
`@app.local_entrypoint()`	Code exécuté localement, point d'entrée
`@modal.fastapi_endpoint()`	Expose une fonction en route HTTP (FastAPI)
`GET`	Récupère une donnée via l'URL
`POST`	Envoie un JSON dans le corps de la requête
`modal.run`	Lance une exécution ponctuelle
`modal.serve`	Démarre un serveur web cloud avec endpoints publics
`docs=True`	Ajoute Swagger UI
`modal.enable_output()`	Affiche les logs cloud dans la console
`app.run()`	Simule un `modal run` localement