Deploying an AI Agent - documentazione

1. LangGraph - Introduzione
2. LangServe - AWS Copilot
3. LangGraph Platform

Introduzione a LangGraph

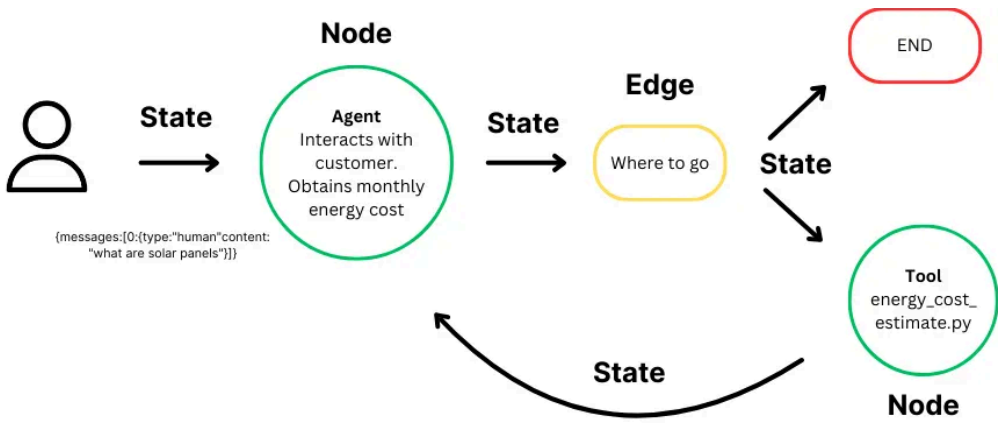
Prima di affrontare l'argomento principale di questa documentazione riguardante il 'deployment' di Agent vorrei fare una breve introduzione a LangGraph.

LangGraph è una libreria concepita come 'estensione', dunque costruita 'on-top', di LangChain.

Mentre LangChain permette lo sviluppo di applicazione LLM via DAGs, ovvero Directed-Acyclic-Graphs, che in termini più semplici significa lo sviluppo lineare di workflows, dunque come 'chaining' di 'strumenti' e 'flussi di lavoro' lineari, LangGraph permette l'aggiunta di 'cicli': ovvero di fare dei 'loop' tra diversi 'stati' e 'nodi' costruiti dentro l'applicazione che ha come 'motore' centrale di tutto un modello LLM.

Questo 'looping' tra 'stati' e 'nodi' permette lo sviluppo di comportamenti complessi, e sopratutto consentono all Agent di decidere 'dinamicamente', dunque basandosi su azioni pregresse o evolversi di date situazioni, che cosa fare successivamente: il risultato è quello che si vuole tipicamente ottenere da un Agent, ovvero una sorta di agire o comportarsi 'ragionato'.

Al 'core' di LangGraph dunque risiede il concetto di 'stateful graph'.
Uno 'stateful graph' è compost di tre elementi principaili:
'state' / 'nodes' / 'edges'.



Ma la domanda tra sviluppatori e tecnici sorge spontanea: è veramente utile questo framework?
La mia risposta è sì, è utile, poichè aggiunge un layer di astrazione piuttosto comodo ad uno sviluppo e orchestrazione già di per sè complicato, quale è lo sviluppo di Agent e LLM application; ma soprattutto mette a disposizione una vasta libreria che si completa e si aggiorna con le API e documentazione di tutti i vari LLM providers: dunque i grandi players - Anthropic, OpenAI, Mistral, DeepSeek, Groq ed anche con i vari tool che mettono a disposizione i cloud-provider AWS, GCP, Alibaba, Azure.

Ovviamente bisogna comprendere che ognuno tira acqua al suo mulino...e infatti era da un pò che stavo aspettando i soliti 'tre' pacchetti per un accedere ad un utilizzo 'avanzato' di LangChain, LangGraph e infine LangSmith: ormai lo sviluppo di software client è da anni volto in questa direzione, se notate anche le UIs di questi progetti-prodotti sono quasi completamente identiche.

Ma di certo non sarò io a biasimarli!
Onestamente preferisco contare su di una libreria utilizzata e condivisa da una community di sviluppatori e mantenuta da programmatori-imprenditori con in mente una direzione di sviluppo, che costruirmi le cose a mano da zero, del resto altrimenti saremmo qua ancora a scrivere le cose in JavaScript senza usare framework con React o Vue o simile.

LangServe - AWS Copilot

Lasciandoci alle spalle considerazioni imprenditoriali ed etiche, vorrei arrivare al nocciolo della questione: il deployment di Agents.

Questa prima modalità per il deployment di Agent si affida all'uso di LangServe, che al momento viene ancora mantenuto, ma ovviamente stanno cercando di dare maggior spazio alla piattaforma LangSmith che comprende tutti i servizi di cloud-services che LangChain e LangGraph mettono a disposizione.

Questa soluzione offre un deployment senza l'utilizzo di una piattaforma ‘centrale’, ma attraverso l’invocazione di diversi strumenti.

Con questa prima modalità potremo fare i deploy della soluzione in una API a cui potremo accedere per aggiungerla a qualsiasi frontend.
I punti principali di questo metodo si basano su:

1. Inizializzare LangServe Application usando il comando:

langchain serve new .

Questo comando serve per creare la struttura di base per l'applicazione LangServe, includendo tutte le directories e file che serviranno per il deploy.

In combinazione a invocare il comando di LangServe, installiamo Poetry, ma questa rappresenta semplicemente una ‘best-practice’ per l’utilizzo di LangServe, per mantenere aggiornato e stabili tutte le varie dipendenza del progetto.

Dopodichè si prosegue facendo l'inizializzazione di FastAI che si occuperà di gestire tutte le richieste in entrata dell AI Agent.

2. FastAPI:

```
# initialize FastAPI
app = FastAPI()
```

Questo comando esegue il setup di FastAPI, utilizzata per interfacciarsi tra gli utenti e l AI Agent.

Testing Locale

Si procede testando localmente l endpoint.
Prima di tutto bisogna assicurarsi di aver creato un ‘DockerFile’, questo perché LangServe sfrutta Docker per il deployment.

Quindi possiamo a tutti gli effetti proseguire con il comando:

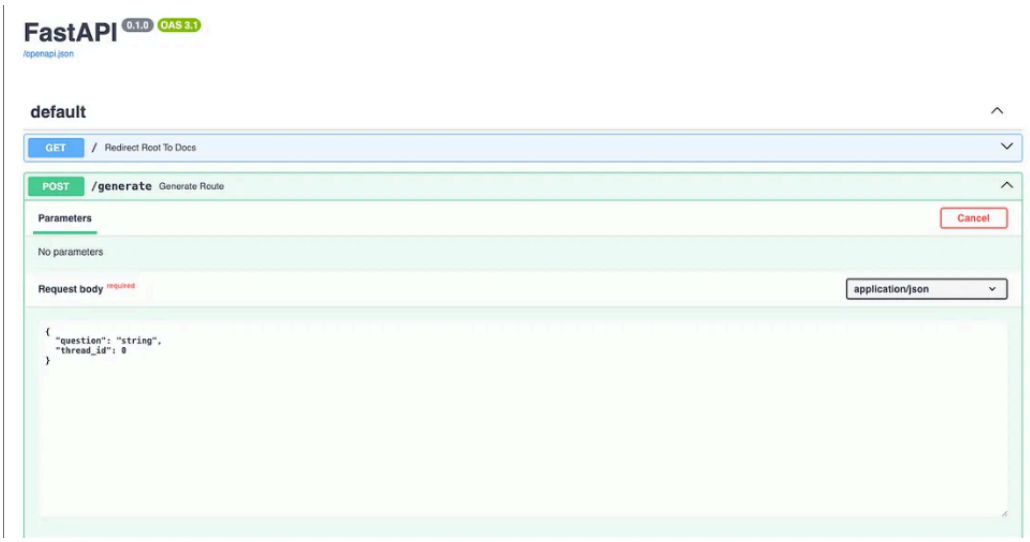
```
# start LangServe application
langchain serve
```

Questo comando avvierà un server locale che ospita l'API del tuo agente AI.
LangServe creerà automaticamente un ambiente usando Docker, e potrai accedere alla tua API da `http://localhost:8000`.

Accesso all API EndPoint

Una volta che il server è in funzione, puoi aprire il tuo browser e visitare `http://localhost:8000/docs`.

Qui vedrai la route `POST /generate`, che viene usata per interagire con l'agente AI. Puoi testare l'API direttamente da questa pagina cliccando sulla route, poi su "Try it out" e compilando i campi richiesti (come la domanda e il thread_id).



Con questa pipeline piuttosto regolare abbiamo testato ‘localmente’ il nostro API EndPoint, e soprattutto abbiamo testato il suo funzionamento!

CloudDeployment con AWS Copilot

Ora è il momento del deployment su Cloud affinché si possa uscire dall’ambiente locale e rendere l AI Agent accessibile pubblicamente.
Useremo **AWS Copilot** per deployare l'agente AI. AWS Copilot semplifica il processo di deployment di applicazioni containerizzate, come il nostro agente AI che gira con LangServe, impostando automaticamente l'infrastruttura necessaria.
Con questo comando inizializziamo AWS Copilot via terminal dentro il progetto:

```
copilot init \
--app ai-agent \
--name ai-agent \
--type 'Load Balanced Web Service' \
--dockerfile './Dockerfile'
```

In questo modo AWS Copilot creerà diversi componenti chiave e l'infrastruttura per il nostro AI Agent:

1. **Elastic Load Balancer (ELB)**: Distribuisce il traffico in entrata tra più istanze del tuo servizio per garantire alta disponibilità e tolleranza ai guasti. Quando il traffico aumenta, ELB può distribuire il carico per evitare che una singola istanza venga sovraccaricata, assicurando prestazioni fluide e scalabilità per il tuo agente AI.
2. **ECS (Elastic Container Service)**: È un servizio di orchestrazione di container completamente gestito che AWS usa per deployare e gestire applicazioni containerizzate. In questo caso, ECS gestirà l'esecuzione, la scalabilità e il monitoraggio del tuo agente AI all'interno dei container Docker. Funziona perfettamente con ELB per garantire che il tuo servizio possa scalare automaticamente in base alla domanda.

Dopo questa configurazione, ci sono alcuni dettagli di AWS Policies ma che ometterò, perchè i casi d'uso sono veramente variegati.

Per effettivamente deployare l agent, basterà questo comando:

```
copilot deploy
```

Questo comando avvierà il deployment della tua applicazione su AWS. Durante questo processo, Copilot:

1. **Costruirà e Caricherà il Container**: La tua applicazione sarà impacchettata come container Docker e caricata su **Amazon Elastic Container Registry (ECR)**.
2. **Avvierà il Servizio**: Copilot creerà l'infrastruttura necessaria per eseguire il tuo servizio su **Amazon Elastic Container Service (ECS)**. ECS gestisce il deployment, la scalabilità e l'esecuzione dell'applicazione containerizzata.
3. **Configurerà Rete e Bilanciamento del Carico**: Copilot configurerà un **Elastic Load Balancer (ELB)** per indirizzare il traffico in entrata al tuo servizio, garantendo alta disponibilità e tolleranza ai guasti.
4. **Assegnerà i Permessi**: Tutte le policy necessarie, come la **Bedrock Access Policy**, saranno associate al ruolo del task ECS, assicurando che il tuo servizio abbia i permessi necessari per interagire con i modelli AWS Bedrock.

Come sempre AWS lascia molto poco al caso, dunque ci fornisce comandi per capire le varie problematiche che possono accadere, con questo comando possiamo accedere ai logs di errore:

```
copilot svc log
```

Una volta completato il deployment, AWS Copilot ti fornirà un URL pubblico per il tuo agente AI.
Possiamo accedere al browser e visitare l'URL per accedere alla documentazione FastAPI all'indirizzo: https://<your-app-url>.aws.com/docs

Qui, vedrai la route **POST /generate**, che ti permette di interagire con l'agente AI. Puoi testare nuovamente l'API direttamente cliccando sulla route, selezionando "Try it out" e compilando i campi richiesti, come `question` e `thread_id`.

Conclusione e considerazione su Multitenancy

In questo momento siamo riusciti a deployare con successo un AI Agent progettato con LangGraph e ‘reso disponibile’ attraverso un mix di servizi esterni.

Il servizio che balza subito all’occhio e che sicuramente svolge il ruolo più importante è AWS Copilot.
Questo servizio messo a disposizione di AWS una volta fatto partire, attraverso delle policies che bisogna configurare, mette in moto una catena di altri servizi: per avere un architettura ‘minimal’ sostanzialmente sono questi due:

- ELB - Elastic Load Balancer - che sostanzialmente è un load-balancer ovvero si occupa di gestire le richieste e il traffico che intercorre nella tua applicazione e tra l’altro fornisce anche un layer di ‘fault tolerance’ che sostanzialmente significa che se qualcosa, in ambito di hosting, ‘fallisce’ il servizio continua ad essere disponibile per gli utenti.
- ECS - Elastic Container Service - è un servizio di orchestration che è a tutti gli effetti dove sta ‘runnando’ la nostra applicazione, come un docker, con la differenza che è su AWS, permette un facile monitoring delle performance e della salute del software ma soprattutto permette un facile ‘scaling’.

In questo momento il nostro AI Agent ha a disposizione questa architettura per essere deployato e disponibile al pubblico.

Ma questa architettura, è considerabile multi-tenant o fully-tenant? Sì e no, dipende.

In questo momento il nostro AI Agent è in un ambiente dockerizzato, ne stiamo monitorando la salute, stiamo guardando le metriche di traffico e possiamo anche facilmente scalarlo per renderlo disponibile ad un pubblico più o meno ampio, ma se vogliamo essere molto specifici, questo non è considerabile un ‘fully-tenant’.

Per prendere una definizione da manuale di multi-tenancy propongo quella che si trova in 'Build Clean Architecture' by William Vance:

"Multitenancy refers to a software architecture where a single instance of an application serves multiple customers (tenants). Each tenant's data is isolated from other tenants, but they all use the same application and infrastructure. This is common in SaaS (Software as a Service) applications."

Che tradotto in parole molto semplici è: implementazione di altri servizi.
Soprattutto se vogliamo restare in ambito di CloudProvider dunque un PaaS - Platform-as-a-Service, si traduce nell'utilizzo di altri servizi per raggiungere un full-multi-tenant.

Soltanto per menzionarli, questi sono i servizi che si dovrebbero integrare:

1. **AWS Cognito**
    - Autenticazione/autorizzazione di utenti e tenant
    - Gestisce pool di utenti e pool di identità
    - Supporta registrazione, accesso e controllo degli accessi
2. **AWS IAM**
    - Gestione dettagliata dei permessi
    - Creazione di ruoli e policy specifici per tenant
    - Controllo dell'accesso alle risorse AWS per tenant
3. **Amazon DynamoDB** o **RDS**
    - Memorizzazione di configurazioni e metadati dei tenant
    - Mantenimento di dati specifici per tenant con chiavi di partizione per l'isolamento
    - Archiviazione della cronologia delle conversazioni e degli stati degli agenti per tenant
4. **Amazon API Gateway**
    - Creazione di chiavi API specifiche per tenant
    - Implementazione della limitazione delle richieste per tenant
    - Impostazione di piani di utilizzo e quote
5. **AWS Lambda**
    - Esecuzione di logiche o personalizzazioni specifiche per tenant
    - Elaborazione delle richieste con contesto del tenant
    - Gestione di integrazioni specifiche per tenant
6. **Parameter Store / AWS Secrets Manager**
    - Archiviazione sicura delle configurazioni specifiche dei tenant
    - Gestione di diverse chiavi API per tenant
7. **AWS CloudWatch**
    - Monitoraggio dei modelli di utilizzo per tenant
    - Configurazione di allarmi specifici per tenant
    - Tracciamento del consumo di risorse per tenant

8. **AWS CloudFront** con **Lambda@Edge**
    - Instradamento delle richieste basato sulle informazioni del tenant
    - Applicazione di strategie di caching specifiche per tenant
9. **Amazon SQS/SNS**
    - Elaborazione di flussi di lavoro asincroni specifici per tenant
    - Gestione delle notifiche per tenant
10. **AWS WAF**
    - Implementazione di regole di sicurezza specifiche per tenant
    - Protezione contro attacchi mirati a specifici tenant

E la pipeline di architettura dovrebbe essere così:

1. Cognito gestisce l'autenticazione
2. API Gateway indirizza le richieste al servizio LangGraph con il contesto del tenant
3. L'applicazione legge la configurazione del tenant da DynamoDB
4. L'agente elabora la richiesta con impostazioni specifiche per tenant
5. I risultati e l'utilizzo vengono registrati su CloudWatch con identificatori del tenant
6. Le informazioni di fatturazione vengono acquisite per tenant

Compromessi tra costi e funzionalità

Per concludere, implementare una soluzione fully-tenant comporta dei trade-off significativi.

Da un lato, l'aggiunta di tutti questi servizi AWS porta a un incremento notevole dei costi - non solo di infrastruttura ma anche di sviluppo e manutenzione.
Ogni servizio aggiuntivo richiede configurazione, monitoraggio e expertise specifiche.
Dall'altro lato, otteniamo funzionalità avanzate come isolamento completo dei dati, personalizzazione per cliente e scalabilità granulare.
La domanda da porsi è: questi costi aggiuntivi sono giustificati dal valore che portano al tuo specifico caso d'uso?

L'architettura attuale (LangServe + AWS Copilot + ECS + ELB) è più che adeguata in diversi scenari: per MVP e proof-of-concept, per applicazioni interne aziendali con un bacino limitato di utenti, o per servizi dove non c'è necessità di personalizzazione specifica per cliente.
È una soluzione ottima quando la velocità di deployment è prioritaria rispetto alla complessità di gestione.

È invece necessario passare a una soluzione completamente multi-tenant quando: serviamo clienti enterprise con requisiti di sicurezza stringenti, abbiamo bisogno di applicare quote e limitazioni diverse per cliente, vogliamo offrire personalizzazioni specifiche dell'AI agent per tenant, o quando la scalabilità per singolo cliente diventa un fattore critico.
In questi casi, l'investimento in una architettura più complessa si traduce in valore tangibile per il business.

LangGraph Platform

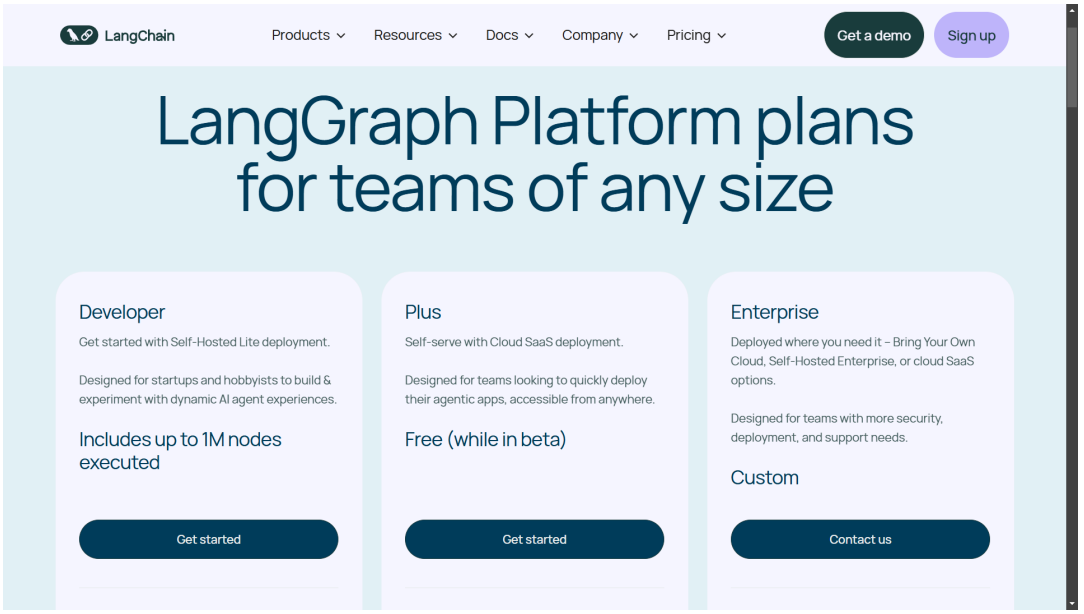
LangGraph Platform offre un'alternativa interessante rispetto all'architettura che abbiamo esplorato finora.
A differenza del setup con AWS Copilot che richiede una configurazione di vari servizi AWS, LangGraph Platform si propone come soluzione più integrata e semplificata per il deployment di agenti AI.

In sostanza, LangGraph Platform fornisce un ambiente già pronto dove deployare i nostri agenti, riducendo significativamente la complessità dell'infrastruttura da gestire.
Questo approccio da PaaS - Platform-as-a-Service - specifico per agenti LangGraph elimina la necessità di configurare manualmente container, load balancer e servizi di orchestration.

I vantaggi principali di questo approccio includono:

- Setup più veloce e con meno configurazione manuale
- Gestione integrata delle dipendenze e versioni
- Monitoraggio e osservabilità già integrati nella piattaforma
- Scalabilità automatica senza dover configurare servizi aggiuntivi

Come avevo già introdotto questa platform si propone offrendo i tre *soliti* pacchetti che ormai quasi tutte le aziende PaaS che propongono SDK si premurano di offrire:



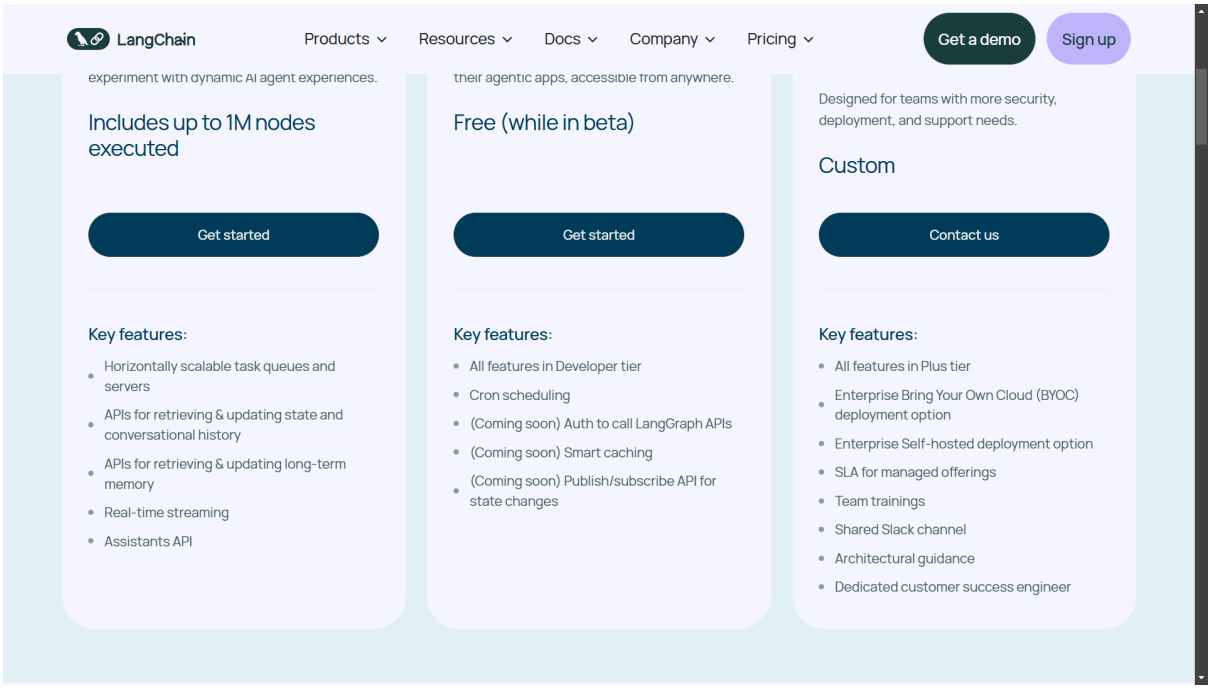
prima di addentrarci nei dettagli dei tre pacchetti proposti, volevo proporre una visione *high-level* più dettagliata dell'offerta di LangGraph Platform che si articola in sei punti chiave:

1. **Motivazione della piattaforma**: Un modo ottimizzato e con opinioni ben definite per deployare e gestire applicazioni LangGraph, riducendo la complessità.

2. **Architettura**: La piattaforma si basa su un'architettura high-level pensata specificamente per applicazioni LangGraph.
3. **Scalabilità e resilienza**: Caratteristiche fondamentali incorporate nel design della piattaforma per garantire stabilità anche con carichi elevati.
4. **Opzioni di deployment**: Quattro diverse modalità di deployment per adattarsi a varie esigenze:
    - Self-Hosted Lite
    - Self-Hosted Enterprise
    - BYOC (Bring Your Own Cloud)
    - Cloud SaaS
5. **Piani disponibili**: Tre diversi livelli di servizio con funzionalità crescenti:
    - Developer: per sviluppo e testing
    - Plus: per implementazioni di media scala
    - Enterprise: per soluzioni aziendali complete
6. **Applicazioni template**: App di riferimento già pronte che fungono da punto di partenza per accelerare lo sviluppo con LangGraph.

Questa struttura offre flessibilità nella scelta del livello di gestione dell'infrastruttura desiderato, permettendo di bilanciare controllo e semplicità in base alle specifiche esigenze del progetto.

Di seguito troviamo le varie *features* dei diversi piani:

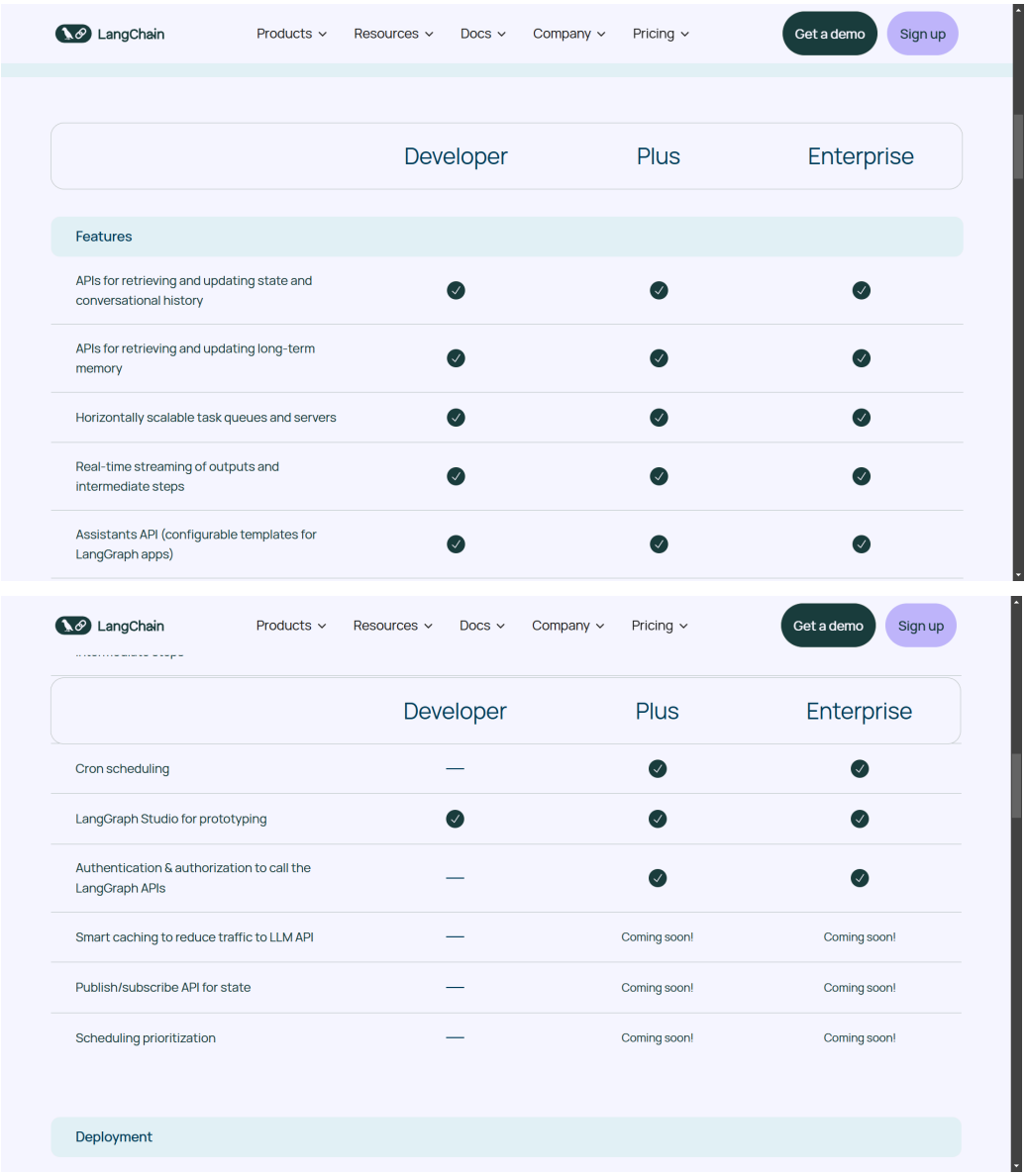


Dunque la domanda sorge di nuovo spontanea, quale piano fa per te?

Copiando e incollando dal sito di LangChain troviamo una mini guida per capire quale piano possa essere più utile in base all'utilizzo:

https://www.langchain.com/pricing-langgraph-platform

Se sei uno sviluppatore individuale che vuole deployare sulla propria infrastruttura, il piano Developer è la scelta perfetta.
Per i team che vogliono utilizzare LangGraph Platform in self-service e deployare sul servizio Cloud gestito, dai un'occhiata al piano Plus.
Se invece hai bisogno di funzionalità avanzate di amministrazione, autenticazione e autorizzazione, opzioni di deployment più flessibili, supporto dedicato o fatturazione annuale, il piano Enterprise è quello giusto per te.



Adesso faremo una breve, per quanto possibile, benchmark di tutte le features che propongono i vari piani, questa volta aiutati da un gentile LLM.
Le features sono divise tra ‘core’, ovvero features disponibili di base per tutti e tre le proposte, avanzate, solamente per i piani ‘plus’ ed ‘enterprise’ e ‘in arrivo’:

Core Features (disponibili in tutti i piani - Developer, Plus, Enterprise):

- **API per stato e cronologia conversazionale**: Permettono di salvare e recuperare lo stato delle conversazioni con gli agenti, fondamentale per mantenere il contesto tra le interazioni.
- **API per memoria a lungo termine**: Gestiscono il salvataggio e il recupero di informazioni persistenti, permettendo agli agenti di "ricordare" informazioni rilevanti nel tempo.
- **Task queue scalabili orizzontalmente**: Gestiscono le richieste in entrata distribuendo su più server, garantendo che il sistema possa gestire carichi elevati senza problemi.
- **Streaming real-time**: Forniscono output in tempo reale durante l'elaborazione, mostrando all'utente i passaggi intermedi (tipo quando vedi l'AI che "pensa" mentre scrive).
- **Assistants API con template configurabili**: Permettono di creare rapidamente interfacce per le app LangGraph con modelli pronti all'uso.
- **LangGraph Studio per prototipazione**: Un ambiente di sviluppo per testare rapidamente i tuoi agent senza dover configurare tutto da zero.

Features avanzate (disponibili solo nei piani Plus e Enterprise):

- **Cron scheduling**: Permette di programmare l'esecuzione di task in momenti specifici, utilissimo per automazioni periodiche.
- **Autenticazione e autorizzazione**: Aggiunge un layer di sicurezza per controllare chi può accedere alle API LangGraph, fondamentale per applicazioni multi-utente.

Features in arrivo (marcate come "Coming soon!"):

- **Smart caching per LLM API**: Ridurrà il traffico verso i modelli linguistici memorizzando risposte comuni, tagliando costi e migliorando performance.
- **API Publish/subscribe per stato**: Probabilmente permetterà di sottoscriversi a cambiamenti di stato specifici, perfetto per applicazioni real-time.
- **Scheduling prioritization**: Permetterà di dare priorità a certi task rispetto ad altri, essenziale per gestire richieste con diversi livelli di urgenza.

Sostanzialmente la differenza principale tra i piani è che Developer offre le funzionalità base per sviluppatori individuali, mentre Plus ed Enterprise aggiungono le caratteristiche necessarie per team e aziende, come autenticazione, scheduling e (prossimamente) ottimizzazioni di performance.

Adesso vorrei fare un benchmark anche delle quattro opzioni di deployment che LangGraph Platform mette a disposizione:

1. Self-Hosted Lite

Disponibile per: Tutti i piani (Developer, Plus, Enterprise)

Questa è la versione base per chi vuole hostare in autonomia. È gratuita fino a 1 milione di nodi eseguiti all'anno, perfetta per iniziare a sperimentare senza costi iniziali. Se scegli questa opzione, sei tu responsabile dell'infrastruttura, inclusi database e istanze Redis necessarie.

Dovrai costruire un'immagine Docker usando la CLI di LangGraph e poi deployarla sulla tua infrastruttura. Un dettaglio importante: i cron job non sono disponibili in questa versione.

La vista Deployments di LangGraph Platform è disponibile opzionalmente anche per i deployment Self-Hosted Lite. Con un solo click, puoi deployare i tuoi agenti LangGraph nello stesso cluster Kubernetes dove hai già installato LangSmith.

2. Self-Hosted Enterprise

Disponibile per: Solo piano Enterprise

Questa è la versione premium per chi vuole hostare in autonomia ma con tutte le funzionalità avanzate. Come per la versione Lite, sei responsabile dell'infrastruttura, database e Redis inclusi.

Anche qui si costruisce un'immagine Docker con la CLI di LangGraph per il deployment, ma avrai accesso a tutte le funzionalità enterprise (autenticazione, scheduling avanzato, ecc.). La vista Deployments di LangGraph Platform è disponibile opzionalmente, permettendoti di deployare con un click nello stesso cluster Kubernetes di LangSmith.

3. Cloud SaaS

Disponibile per: Piani Plus ed Enterprise

Se non vuoi gestire infrastrutture, questa è l'opzione per te.
Il servizio Cloud SaaS di LangGraph Platform è hostato direttamente all'interno di LangSmith, offrendo un modo semplice e veloce per deployare e gestire le tue applicazioni LangGraph.

Questa opzione ti dà accesso all'interfaccia UI di LangGraph Platform (integrata in LangSmith) e un'integrazione con GitHub che ti permette di deployare codice direttamente dai tuoi repository. Tutto è gestito dal team di LangChain, quindi zero sbattimento per te.

4. Bring Your Own Cloud (BYOC)

Disponibile per: Solo piano Enterprise e **solo su AWS**

Questa opzione combina il meglio di entrambi i mondi: la semplicità del Cloud con la sicurezza del Self-Hosted. Crei i tuoi deployment attraverso l'UI di LangGraph Platform (in LangSmith) ma l'infrastruttura gira completamente nel tuo cloud AWS.

Il team di LangChain gestisce l'infrastruttura, quindi non devi preoccuparti della manutenzione, ma hai il vantaggio che tutto rimane nel tuo ambiente cloud, garantendo maggiore controllo e conformità alle policy aziendali.

Pricing

Il pricing di LangGraph Platform è strutturato in modo diverso, ma non è chiarissimo per quanto mi riguarda:

- **Piano Developer**: Principalmente gratuito se self-hosted, con limite di 1 milione di nodi eseguiti/anno. Oltre questo limite, presumibilmente ci sono costi aggiuntivi non ben specificati nella documentazione.
- **Piano Plus**: Orientato a team, ma la struttura di prezzo non è trasparente dal sito. Probabilmente include:
    - Quota base mensile per l'accesso alla piattaforma
    - Costi aggiuntivi basati sull'utilizzo (nodi eseguiti, storage, ecc.)
    - Il deployment su Cloud SaaS potrebbe avere costi diversi rispetto al self-hosted
- **Piano Enterprise**: Tipicamente con pricing personalizzato basato su contrattazione diretta, include supporto premium e funzionalità avanzate.

Conclusione

Come spesso accade con le soluzioni PaaS, questo comporta alcuni trade-off in termini di flessibilità. Con AWS Copilot abbiamo maggiore controllo sull'infrastruttura sottostante e possiamo personalizzare ogni aspetto dell'architettura, mentre con LangGraph Platform accettiamo alcune scelte predefinite in cambio di maggiore semplicità.

Per quanto riguarda la multi-tenancy, anche LangGraph Platform richiederà integrazioni con servizi esterni per gestire autenticazione, isolamento dei dati e personalizzazioni per tenant, ma pffre alcune funzionalità di base già integrate nella piattaforma.

In definitiva, la scelta tra AWS Copilot e LangGraph Platform dipenderà dalle specifiche esigenze del progetto: se prioritizziamo velocità di deployment e semplicità di gestione, LangGraph Platform potrebbe essere la scelta migliore; se invece abbiamo requisiti specifici di personalizzazione dell'infrastruttura o integrazione con altri servizi AWS, il setup con AWS Copilot rimane più appropriato.

Adesso concediamoci qualche riflessione su i diversi piani proposti dall’azienda:

Il piano Developer è sostanzialmente progettato per sviluppatori individuali o piccoli team che vogliono sperimentare.
Offre le funzionalità core che consentono di costruire e testare agenti AI funzionali - come le API per gestire stato e memoria, lo streaming real-time e le code scalabili. Questo è sufficiente per la prototipazione e lo sviluppo iniziale, ma manca di caratteristiche essenziali per un ambiente di produzione.

I piani Plus ed Enterprise aggiungono elementi cruciali per un'implementazione seria: autenticazione/autorizzazione e scheduling. Questa non è una scelta casuale - LangChain sta essenzialmente dicendo "se vuoi mettere questo in produzione con utenti reali, avrai bisogno di queste funzionalità", e hanno ragione. La mancanza di autenticazione nel piano Developer è particolarmente significativa e limita fortemente il suo utilizzo in scenari reali.

Le funzionalità "coming soon" sono particolarmente interessanti perché mostrano la direzione di evoluzione della piattaforma. Lo smart caching per le API LLM è una caratteristica potenzialmente risolutiva per due problemi enormi negli agenti AI: latenza e costi. Se implementato correttamente, potrebbe ridurre drasticamente entrambi.

Confrontando questa offerta con l'approccio AWS Copilot che abbiamo esaminato prima, vedo vantaggi e svantaggi. LangGraph Platform offre un'esperienza più integrata e specifica per agenti AI, con meno configurazione e gestione.
AWS Copilot richiede più lavoro di setup ma offre maggiore flessibilità e controllo.

La domanda come sempre, sorge spontanea, questa platform è utile?
Sì, questa piattaforma è chiaramente progettata per semplificare il deployment di agenti AI, ma con un evidente modello di business che spinge verso l'upgrade ai piani superiori quando si vuole passare dalla fase di sviluppo a quella di produzione.
Per progetti piccoli o MVP, il piano Developer potrebbe essere un ottimo punto di partenza, ma per qualsiasi cosa da production, Plus diventa praticamente obbligatorio.

Un altro aspetto da considerare è la dipendenza tecnologica: affidarsi a LangGraph Platform significa legarsi al loro ecosistema.
Questo è il classico trade-off comodità vs controllo che vediamo in molte soluzioni PaaS. Se stai costruendo qualcosa di critico per il business, dovresti valutare attentamente quanto controllo sei disposto a cedere in cambio della semplicità.

In definitiva, questa piattaforma sembra particolarmente adatta a team che vogliono velocizzare il time-to-market di applicazioni basate su agenti AI senza doversi preoccupare troppo dell'infrastruttura sottostante.
Per chi ha requisiti particolari di personalizzazione o integrazioni complesse, l'approccio AWS Copilot potrebbe rimanere più adeguato, nonostante richieda più lavoro iniziale.