Guida a Amazon Bedrock: Portare i Foundation Model in produzione

22 Aprile 2026 - 7 min. read

Amazon Bedrock | Foundation Models | Generative AI | Large Language Models | Machine Learning

Introduzione

Generative AI, Machine Learning, Large Language Models, Foundation Models.

Se lavorate nel Cloud, questi termini risuonano ovunque: dalle conferenze tech ai meeting aziendali, passando per ogni newsletter che vi arriva in mail. Ma quante volte, dopo l'entusiasmo iniziale, vi siete trovati davanti alla domanda più scomoda: "Ok, ma come lo portiamo in produzione?"

È proprio qui che molti progetti di AI generativa si arenano. Tra modelli da scegliere, infrastrutture da configurare, costi da ottimizzare e vincoli di sicurezza da rispettare, il gap tra il proof-of-concept e la produzione può cambiare notevolmente e rappresentare un blocker al go live del progetto.

Amazon Bedrock nasce esattamente per cercare di risolvere queste problematiche. Non è l'ennesimo servizio che promette miracoli, ma una piattaforma Serverless che rende accessibili i Foundation Models (FM) più avanzati attraverso API gestite, permettendovi di concentrarvi sul **valore di business invece che sull'infrastruttura.** In questa guida, vi mostreremo come utilizzare Amazon Bedrock in scenari reali, dal deploy all'ottimizzazione dei costi, passando per pattern architetturali come RAG e Agents. Perché una cosa è fare un "hello world" con ChatGPT, un'altra è **costruire sistemi production-ready scalabili ed efficienti**.

Cos'è Amazon Bedrock e perché dovreste considerarlo

Amazon Bedrock è un **servizio fully managed** che offre accesso a *Foundation Models* di diverse aziende **leader nel settore AI** attraverso una singola API. Pensatelo come un marketplace di modelli AI dove potete scegliere quello più adatto al vostro use-case senza dovervi preoccupare di training, hosting o scaling.

I Foundation Models disponibili

Bedrock mette a disposizione modelli proprietari ed open-source, tra i quali:

- **Anthropic** (Claude Opus, Sonnet, Haiku)
- **Meta** (Llama)
- **Amazon** (Nova Lite, Nova Pro,Titan Embeddings)
- **AI21 Labs** (Jurassic-2)
- **DeepSeek**
- **MoonShot Kimi**

La varietà non è casuale: **ogni modello ha caratteristiche specifiche in termini di performance, costi, capacità e velocità**. Scegliere quello giusto fa la differenza tra un progetto sostenibile e una bolletta AWS fuori controllo.

Perché Bedrock invece di altre soluzioni?

"Ma non posso usare direttamente le API di OpenAI o hostare un modello open source su EC2?"

Ci sentiamo spesso chiedere.

Certo che potete. Ma è importante considerare questi aspetti:

- **Integrazione nativa AWS**: Bedrock si integra perfettamente con il resto dell'ecosistema AWS. IAM policies, VPC endpoints, CloudWatch metrics, AWS PrivateLink - tutto funziona out of the box. Se la vostra infrastruttura è già su AWS, l'overhead di integrazione è minimo.
- **Gestione semplificata**: Niente server da mantenere, niente modelli da aggiornare manualmente, niente preoccupazioni su availability zones o failover. È Serverless nel vero senso del termine.

- **Data residency e compliance**: I vostri dati non lasciano il vostro account AWS. Per settori regolamentati come finance o healthcare, questo può essere un requirement non negoziabile.
- **Pricing trasparente**: Pay-per-use basato su token processati. Niente commitment iniziali, niente costi nascosti di infrastruttura.

Ma attenzione: come sempre nel cloud, "Serverless" non significa che fa tutto da solo, ma è fondamentale architettare con intelligenza per ottimizzare costi e performance.

Architettura di base: come funziona Bedrock

Capiamo come funziona Bedrock sotto il cofano.

Il modello di invocazione

Bedrock offre principalmente due modalità di utilizzo:

1. **On-demand inference**: Fate una chiamata API, ricevete una risposta. Simple as that. Pagate per quello che usate, **ideale per carichi variabili.**
2. **Provisioned throughput**: Riservate una capacità dedicata per avere latenza prevedibile e costi ottimizzati su volumi elevati. Pensate a **Reserved Instances**, ma per l'AI.

La scelta tra le due dipende, come al solito, dal vostro use-case. Se, per esempio, avete un chatbot con picchi imprevedibili, on-demand è la scelta giusta. Se processate migliaia di documenti al giorno con un volume costante, provisioned throughput può farvi risparmiare fino al 50%.

Componenti chiave dell'architettura

Un'implementazione Bedrock tipica include:

- **Model invocation**: Le chiamate dirette al Foundation Model
- **Knowledge Bases**: Repository vettoriali per implementare RAG
- **Agents**: Orchestratori autonomi che possono usare tool e API esterne
- **Guardrails**: Policy per filtrare contenuti problematici o PII
- **Custom models**: Fine-tuning dei modelli base (quando on-demand non basta)

Nelle prossime sezioni, vedremo ognuno di questi componenti con esempi pratici.

Deploy

Bene la teoria è stata fatta. Ora vediamo come portare Bedrock in produzione usando Infrastructure as Code.

Prerequisiti AWS

Prima di iniziare, assicuratevi di avere:

- Un account AWS con permessi sufficienti
- AWS CLI configurata
- Abilitazione dei modelli desiderati nella console Bedrock (alcuni richiedono esplicita richiesta)

Importante: Non tutti i modelli sono disponibili in tutte le region. Va verificata sempre la disponibilità regionale prima di progettare l'architettura.

Una volta fatto deploy, possiamo usare il nostro foundational model per i nostri workload, un esempio tipico è il RAG.

Integrare Bedrock con un'applicazione esistente

Può capitare che si abbia a che fare con una applicazione già scritta che non utilizza le API native di Amazon Bedrock. Non è affatto un problema: durante il re:Invent 2025, è stato annunciato **Project Mantle**.

Si tratta di un motore di inferenza distribuita progettato per offrire endpoint API **OpenAI-compatibili**. L'idea è estremamente semplice: funge da *drop-in replacement*, permettendo agli sviluppatori di migrare le applicazioni esistenti (nate usando il set di API OpenAI) su Bedrock, semplicemente cambiando l'endpoint dell'API e generando una nuova chiave dalla console AWS.In questo modo si possono usare i modelli presenti su Bedrock senza modificare il codice applicativo, riducendo a zero il tempo di porting delle applicazione. Questa pagina contiene tutto quel che serve per iniziare!

Costi e ottimizzazione: far quadrare i conti

Parliamo di soldi. Perché un chatbot che costa €10,000 al mese per servire 1000 utenti non è sostenibile.

Modello di pricing Bedrock

Bedrock usa un modello pay-per-use basato su:

1. **Input tokens**: Testo che inviate al modello (prompt + contesto)
2. **Output tokens**: Testo generato dal modello
3. **Storage** (per Knowledge Bases): Costo di OpenSearch Serverless o altro vector DB
4. **Provisioned throughput** (opzionale): Capacità riservata

I prezzi variano enormemente tra modelli. Esempio (us-east-1, gennaio 2026):

- **Claude Haiku**: $1/1M input tokens, $5/1M output tokens
- **Claude Sonnet**: $3/1M input tokens, $15/1M output tokens
- **Claude Opus**: $5/1M input tokens, $25/1M output tokens

Usare Opus invece di Haiku per un task semplice può costare **5x di più**. La scelta del modello giusto è fondamentale.

Strategie di ottimizzazione dei costi

1. Model selection intelligente

Non usate sempre il modello più grande. Create una strategia a livelli:

```
`# model_selector.py
def select_model_for_task(task_type, complexity, context_length):
""" Seleziona il modello ottimale in base al task """
if task_type == 'classification' or task_type == 'extraction':
# Task strutturati: Haiku è sufficiente
    return 'anthropic.claude-4-haiku-v1:0'

elif task_type == 'summarization':
    if context_length < 10000:
        return 'anthropic.claude-4-v1:0'
    else:
        # Contesto lungo: Sonnet gestisce meglio
        return 'anthropic.claude-4-v1:0'
```

```
elif task_type == 'reasoning' or complexity == 'high':
    # Ragionamento complesso: Sonnet o Opus
    if context_length > 50000:
        return 'anthropic.claude-4-opus-v1:0'
    else:
        return 'anthropic.claude-4-sonnet-v1:0'

elif task_type == 'code_generation':
    # Sonnet è ottimo per il codice
    return 'anthropic.claude-4-sonnet-v1:0'

else:
    # Default: Haiku per costo minimo
    return 'anthropic.claude-4-haiku-v1:0'
```

Monitoring e cost alerts

Implementate monitoring fin dal giorno uno:

```
# cost_monitoring.py
import boto3
from datetime import datetime, timedelta

cloudwatch = boto3.client('cloudwatch')


def put_cost_metric(model_id, tokens_used, cost):
    """
    Pubblica metriche di costo su CloudWatch
    """
    cloudwatch.put_metric_data(
        Namespace='Bedrock/Usage',
        MetricData=[
            {
                'MetricName': 'TokensUsed',
                'Value': tokens_used,
                'Unit': 'Count',
                'Timestamp': datetime.utcnow(),
```

```
                'Dimensions': [
                    {'Name': 'ModelId', 'Value': model_id}
                ]
            },
            {
                'MetricName': 'EstimatedCost',
                'Value': cost,
                'Unit': 'None',
                'Timestamp': datetime.utcnow(),
                'Dimensions': [
                    {'Name': 'ModelId', 'Value': model_id}
                ]
            }
        ]
    )
```

A questo punto con questa metrica si può creare un allarme CloudWatch sull'utilizzo dei token per trovare anomalie ed evitare costi imprevisti.

Security e Governance: proteggere i dati

Bedrock gestisce informazioni potenzialmente sensibili. La security non è opzionale.

Guardrails: il filtro automatico

Bedrock Guardrails vi permette di filtrare contenuti problematici automaticamente, sia negli input degli utenti che nelle risposte del modello, indipendentemente dal modello utilizzato.

Funziona su sei categorie di policy: content filters, denied topics, word filters, sensitive information filters, contextual grounding check e Automated Reasoning checks. Ognuna è configurabile in modo indipendente, così potete costruire esattamente il livello di protezione che serve.

I content filter coprono sei categorie predefinite di contenuto dannoso: Hate, Insults, Sexual, Violence, Misconduct e Prompt Attack. Per ognuna potete regolare l'intensità del filtro e non è una scelta binaria: potete scegliere in base al contesto dell'applicazione.

Per i dati sensibili, potete scegliere da un elenco predefinito di tipi di PII oppure definire pattern personalizzati tramite espressioni regolari. Utile soprattutto in ambiti regolamentati come finance o healthcare.

Conclusione: da MVP a produzione

Abbiamo coperto molto terreno: deployment infrastructure-as-code, Agents per task automation, ottimizzazione costi, security best practices.

Portare Foundation Models in produzione non è un progetto tecnologico, è un progetto di business. La tecnologia è il mezzo, non il fine.

Prima di scrivere la prima riga di codice, domandatevi:

- Qual è il valore di business misurabile? "Avere un chatbot" non è una risposta.
- Quale metrica di successo useremo? Costo per interazione? Customer satisfaction? Time-to-resolution?
- Abbiamo i dati giusti? Un RAG senza buona documentation è inutile quanto un'auto senza benzina.
- I costi sono sostenibili a scala? €100/mese per un POC sono accettabili. €10,000/mese per 1000 utenti probabilmente no.

Amazon Bedrock rende la tecnologia accessibile. Ma costruire sistemi AI di successo richiede ancora pianificazione, architettura intelligente e continua ottimizzazione.

La buona notizia? Ora avete gli strumenti per farlo.

Nei prossimi articoli approfondiremo use case specifici: come costruire un sistema RAG multi-lingue, come ottimizzare un Agent per ridurre le allucinazioni, come implementare A/B testing tra modelli diversi.

Stay tuned su **#Proud2beCloud**!

Damiano Giorgi

Ex sistemista on-prem, pigro e incline all'automazione di task noiosi. Alla ricerca costante di novità tecnologiche e quindi passato al cloud per trovare nuovi stimoli.L'unico hardware a cui mi dedico ora è quello del mio basso; se non mi trovate in ufficio o in sala prove provate al pub o in qualche aeroporto!
