

# AWS LLM Deployment - Complete Guide

---

## AWS Services Landscape

### Quick Service Reference

Service	Purpose	Use Case
<b>Bedrock</b>	Pre-trained models	Easy, managed, expensive
<b>SageMaker</b>	ML ops platform	Production ML workloads
<b>EC2</b>	Virtual machines	Self-hosted with full control
<b>Lambda</b>	Serverless functions	Async processing
<b>ECS/EKS</b>	Container orchestration	Containerized deployment
<b>S3</b>	Object storage	Model files, configs
<b>RDS</b>	Relational database	Store conversations
<b>ElastiCache</b>	In-memory cache	Cache responses
<b>CloudFront</b>	CDN	Distribute assets

---

## Three Ways to Deploy LLMs on AWS

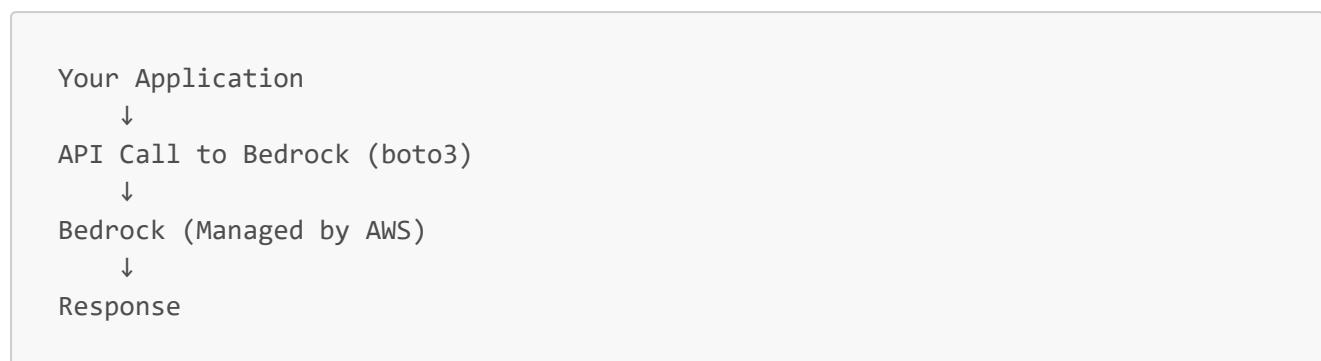
Option 1: AWS Bedrock (Recommended for Beginners)

**What it is:** Managed LLM service - Amazon handles everything

### Supported Models:

- Claude 3 (Anthropic)
- Llama 2 (Meta)
- Mistral 7B (Mistral)
- Jurassic-2 (AI21)

### Architecture:



## Pros:

- No infrastructure management
- Built-in auto-scaling
- Pay per token (no idle cost)
- Easy integration
- High availability built-in

## Cons:

- Most expensive per token
- Limited to Bedrock models
- Can't customize model behavior extensively

## Cost Estimate:

Claude 3 Sonnet:

- Input: \$0.003 per 1K tokens
- Output: \$0.015 per 1K tokens

Example: 1M tokens input, 100K output

$$\text{Cost} = (1,000 \times \$0.003) + (100 \times \$0.015) = \$4.50$$

## When to use:

- Prototyping
- Low-to-medium volume
- Don't want infrastructure management
- Need quick time-to-market

## Quick Start Code:

```
import boto3

client = boto3.client('bedrock-runtime', region_name='us-east-1')

response = client.invoke_model(
    modelId='anthropic.claude-3-sonnet-20240229-v1:0',
    body=json.dumps({
        "max_tokens": 1024,
        "messages": [
            {"role": "user", "content": "What is DevOps?"}
        ]
    })
)

result = json.loads(response['body'].read())
print(result['content'][0]['text'])
```

## **Setup Steps:**

1. AWS Account with billing enabled
  2. Request Bedrock model access (AWS Console → Bedrock → Model Access)
  3. Create IAM role with `bedrock:InvokeModel` permission
  4. Write application code
  5. Deploy (Lambda, ECS, EC2, etc.)
- 

## Option 2: SageMaker (Best for Production)

**What it is:** Amazon's ML operations platform with built-in LLM support

### **Two deployment options:**

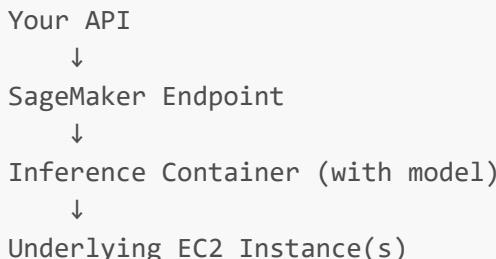
#### **2A. SageMaker JumpStart (Easier)**

- Pre-configured LLM endpoints
- 1-click deployment
- Less control

#### **2B. SageMaker Endpoints (More Control)**

- Custom inference code
- Full control over model
- Better for open-source models

### **Architecture:**



### **Supported Models:**

- Llama 2 (all sizes: 7B, 13B, 70B)
- Mistral 7B
- Falcon
- Custom models

### **Pros:**

- Full control over hardware
- Multi-model endpoints possible

- Good performance
- Kubernetes integration (EKS)
- A/B testing built-in
- Best for production

#### **Cons:**

- More complex setup
- Pay for instance hours (even idle)
- Steeper learning curve
- More infrastructure management

#### **Cost Estimate:**

Llama 2 70B on ml.g5.12xlarge (4 A10G GPUs):

- On-demand: \$7.09/hour
- With 50% utilization: \$3.50/hour for 720 hours = \$2,520/month
- With 80% utilization: \$5.67/hour = \$4,082/month

Much cheaper than Bedrock at scale

#### **When to use:**

- High-volume production
- Need cost optimization
- Multiple models needed
- Custom inference code required

#### **Quick Start Code:**

```
import sagemaker
import json

# Create endpoint (usually done via console or infrastructure as code)
endpoint_name = "llama2-endpoint"

client = sagemaker.client.SageMakerRuntime('us-east-1')

response = client.invoke_endpoint(
    EndpointName=endpoint_name,
    ContentType='application/json',
    Body=json.dumps({
        'inputs': 'What is DevOps?',
        'parameters': {'max_new_tokens': 256}
    })
)
```

```
result = json.loads(response['Body'].read().decode())
print(result)
```

## Setup Steps:

### 1. Using Console (Easiest):

- AWS Console → SageMaker → JumpStart
- Select Llama 2 7B
- Click "Deploy"
- Wait 10 minutes

### 2. Using Infrastructure as Code:

```
# Terraform example
terraform init
terraform apply
```

### 3. Using AWS CLI:

```
aws sagemaker create-endpoint \
--endpoint-name llama2-endpoint \
--endpoint-config-name llama2-config
```

---

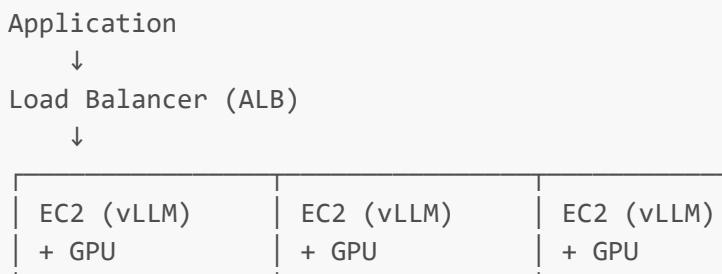
## Option 3: Self-Hosted on EC2 (Most Control, Most Work)

**What it is:** Run LLM on your own EC2 instances with your choice of inference framework

### Popular Inference Frameworks:

- **vLLM:** Fast, high-throughput
- **Ollama:** Simple, great for experimentation
- **Text Generation WebUI:** Web interface for testing
- **LM Studio:** Desktop app, also works on servers

### Architecture:





RDS Database + ElastiCache

### Supported Models:

- Any open-source model (Llama, Mistral, Falcon, etc.)
- Quantized models (smaller, faster, less VRAM)
- Proprietary models (if you have access)

### Pros:

- Most cost-effective at scale
- Unlimited customization
- Run any model
- Can optimize for specific hardware

### Cons:

- X Most complex setup
- X You manage everything
- X Pay for instance hours even if not used
- X Scaling is manual
- X Need to manage GPU drivers, updates

### Cost Estimate:

Llama 2 7B on g4dn.xlarge (1 T4 GPU):

- On-demand: \$0.55/hour
- Reserved (1 year): \$0.35/hour
- 720 hours/month = \$252/month (reserved)

Llama 2 70B on g4dn.12xlarge (4 T4 GPUs):

- On-demand: \$6.48/hour
- Reserved (1 year): \$4.12/hour
- 720 hours/month = \$2,966/month (reserved)

Much cheaper than Bedrock/SageMaker for high volume

### When to use:

- High-volume production (1000+ req/hour)
- Need advanced optimization
- Already have cloud infrastructure team
- Cost optimization critical

### Quick Setup with vLLM:

```

# 1. Launch EC2 instance
# Instance type: g4dn.xlarge (1 T4 GPU) or larger
# AMI: Ubuntu 22.04 LTS
# Storage: 100GB EBS

# 2. SSH into instance
ssh -i your-key.pem ubuntu@instance-ip

# 3. Install dependencies
sudo apt update && sudo apt install python3-pip
pip install vllm

# 4. Run model
python -m vllm.entrypoints.openai.api_server \
--model meta-llama/Llama-2-7b-hf \
--gpu-memory-utilization 0.9

# 5. Test it
curl http://localhost:8000/v1/completions \
-H "Content-Type: application/json" \
-d '{
  "model": "meta-llama/Llama-2-7b-hf",
  "prompt": "What is DevOps?",
  "max_tokens": 256
}'

```

### Application Code:

```

import openai

openai.api_base = "http://ec2-instance-ip:8000/v1"
openai.api_key = "any-string"

response = openai.Completion.create(
    model="meta-llama/Llama-2-7b-hf",
    prompt="What is DevOps?",
    max_tokens=256
)

print(response.choices[0].text)

```

---

## Comparison Table: All Three Options

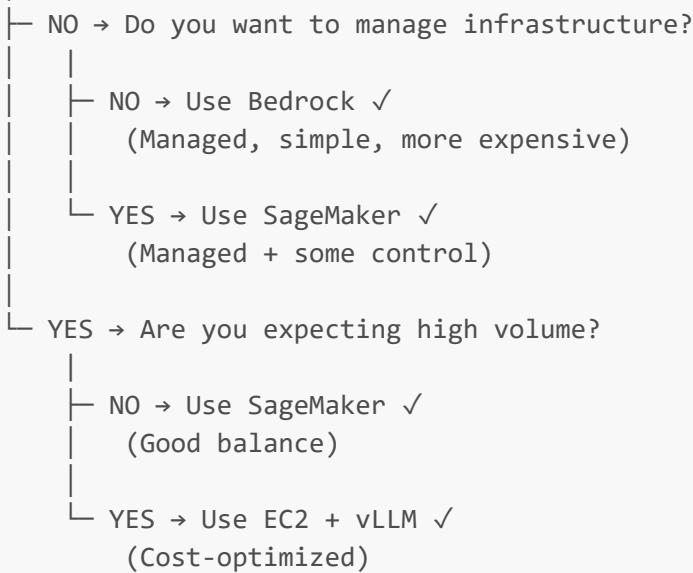
Factor	Bedrock	SageMaker	Self-Hosted EC2
Setup Time	Hours	Days	Weeks

Factor	Bedrock	SageMaker	Self-Hosted EC2
<b>Monthly Cost (1M tokens)</b>	\$4-50	\$500-5000	\$200-3000
<b>Control Level</b>	Low	Medium	High
<b>Ops Overhead</b>	Minimal	Moderate	High
<b>Auto-scaling</b>	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes	<input type="checkbox"/> Manual
<b>Multi-model</b>	<input type="checkbox"/> No	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
<b>Custom models</b>	<input type="checkbox"/> No	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes
<b>Best For</b>	Quick start	Production	High-volume

## Choosing Your Option

### Quick Decision Tree

Start here: Do you have ML expertise?



### Cost-Based Decision

Low volume (< 10K requests/month):  
→ Bedrock (Simplest, per-token pricing)

Medium volume (10K-1M requests/month):  
→ SageMaker (Good balance)

High volume (> 1M requests/month):  
→ Self-hosted EC2 (Most cost-effective)

# Implementation Comparison

## Implementation 1: Bedrock (5 minutes)

```
# app.py
import boto3
from flask import Flask, request, jsonify

app = Flask(__name__)
client = boto3.client('bedrock-runtime')

@app.route('/chat', methods=['POST'])
def chat():
    data = request.json

    response = client.invoke_model(
        modelId='anthropic.claude-3-sonnet-20240229-v1:0',
        body=json.dumps({
            "max_tokens": 1024,
            "messages": [{"role": "user", "content": data['prompt']}]
        })
    )

    result = json.loads(response['body'].read())
    return jsonify({"response": result['content'][0]['text']})

if __name__ == '__main__':
    app.run(port=5000)
```

## Deploy to Lambda or ECS in minutes

## Implementation 2: SageMaker (30 minutes)

```
# First deploy model to endpoint (AWS Console or Terraform)
# Then in your application:

import sagemaker
import json

endpoint_name = "llama2-endpoint"
client = sagemaker.client.SageMakerRuntime()

@app.route('/chat', methods=['POST'])
def chat():
    data = request.json

    response = client.invoke_endpoint(
```

```

        EndpointName=endpoint_name,
        ContentType='application/json',
        Body=json.dumps({
            'inputs': data['prompt'],
            'parameters': {'max_new_tokens': 256}
        })
    )

    result = json.loads(response['Body'].read().decode())
    return jsonify({"response": result})

```

## Plus infrastructure setup (Terraform/CloudFormation)

---

Implementation 3: EC2 + vLLM (1-2 hours)

```

# EC2 user data script
#!/bin/bash
sudo apt update
sudo apt install -y python3-pip
pip install vllm

# Download model
python -m vllm.entrypoints.openai.api_server \
--model meta-llama/Llama-2-7b-hf \
--gpu-memory-utilization 0.9 &

# Application hits local API

```

## Plus load balancer setup (Terraform/CloudFormation)

---

## Next Steps

Choose your path:

1. **Going with Bedrock?** → See [01-Bedrock-Quick-Start.md](#)
  2. **Going with SageMaker?** → See [02-SageMaker-Production.md](#)
  3. **Going with EC2?** → See [03-EC2-vLLM-Setup.md](#)
  4. **Comparing costs?** → Jump to [06-Cost-Optimization/](#)
- 

## AWS IAM Permissions Needed

For Bedrock:

```
{
    "Version": "2012-10-17",
```

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "bedrock:InvokeModel"
    ],
    "Resource": "arn:aws:bedrock:*:ACCOUNT:foundation-model/*"
  }
]
```

For SageMaker:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sagemaker:InvokeEndpoint",
        "sagemaker:DescribeEndpoint"
      ],
      "Resource": "arn:aws:sagemaker:*:ACCOUNT:endpoint/*"
    }
  ]
}
```

For EC2:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:)"
      ],
      "Resource": "*"
    }
  ]
}
```

---

**Key Takeaway:** Start with Bedrock for simplicity, move to SageMaker for production balance, scale to EC2 for cost efficiency.

---