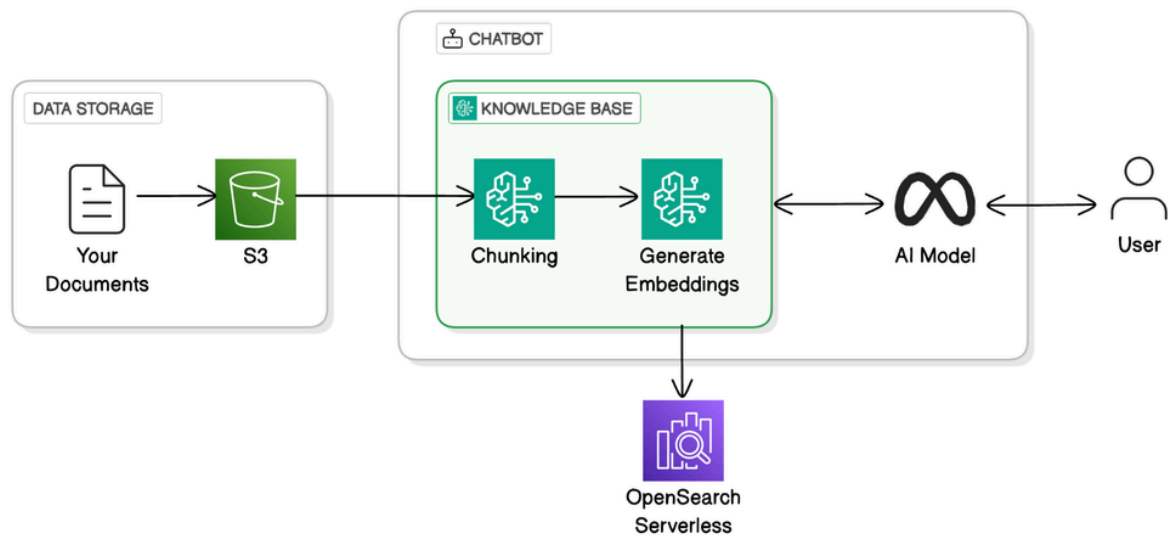




Kareshma  
Rajaananthapadmanaban

# Chat With Your Bot in the Terminal

Build a RAG chatbot that gives you answers straight  
from the terminal





nextwork.org

# Interacting With My RAG Chatbot in the Terminal



Kareshma Rajaananthapadmanaban

```
CloudShell
us-east-2 +
~ $ aws bedrock-agent-runtime retrieve-and-generate \
> --input '{"text": "What is NextWork?"}' \
> --retrieve-and-generate-configuration '{
>   "knowledgeBaseConfiguration": {
>     "knowledgeBaseId": "ZI0PVFJXBH",
>     "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0"
>   },
>   "type": "KNOWLEDGE_BASE"
> }'
> {
>   "citations": [
>     {
>       "generatedResponsePart": {
>         "textResponsePart": {
>           "span": {
>             "end": 182,
>             "start": 0
>           },
>           "text": "Nextwork is a website that offers various projects, including those related to AWS DevOps and Terrafor
m, and has a community forum where users can ask questions and share information."
>         }
>       },
>       "retrievedReferences": [
>         {
>           "content": {
>             "text": "https://community.nextwork.org/c/i-have-a-question?automatic_login=true https://community.
nextwork.org/c/i-have-a-question?automatic_login=true https://community.nextwork.org/c/i-have-a-question?automatic_login-tr
ue
>             Website loaded through CloudFront. To improve performance and security, I enabled caching and HTTPS in CloudFront. This ensur
es that static assets are served quickly and securely, even during high traffic. Updating the site is simple just re-upload files t
o the S3 bucket, and CloudFront handles the global delivery with minimal delay. https://community.nextwork.org/c/i-have-a
-question?automatic_login=true https://community.nextwork.org/c/i-have-a-question?automatic_login=true https://comm
unity.nextwork.org/c/i-have-a-question?automatic_login=true https://community.nextwork.org/c/i-have-a-question?automatic_lo
gin=trueKareshma Rajaananthapadmanaban nex t wor k.org Logic tier To build the logic tier, I will use AWS Lambda to run Bac
kend code in response to HTTP requests managed by API Gateway. This eliminates the need to manage servers and enables scalable exec
ution of application logic whenever a user triggers an action from the frontend. The Lambda function retrieves data by using th
e AWS SDK to connect to DynamoDB."
>           "type": "TEXT"
>         },
>         {
>           "location": {
>             "s3Location": {
>               "uri": "s3://nextwork-rag-bedrock-kareshma/Three-tier Web app.pdf"
>             }
>           },
>           "type": "S3"
>         }
>       ],
>       "metadata": {
```



# Introducing Today's Project!

In this project, I will build a **RAG chatbot** that answers questions directly from the **terminal** using Amazon Bedrock. I'm doing this project to learn how to **interact** with my **chatbot** through the **AWS CLI** in CloudShell, manage a Knowledge Base without relying on the console, and practice command-line workflows. This helps me gain practical skills useful for DevOps and backend engineering, where speed, automation, and control matter.

## Tools and concepts

The services I used were **Amazon Bedrock, S3, CloudShell**, and the **Bedrock Agent/Runtime**. The key concepts I learnt included creating and uploading files to S3, syncing and updating a Knowledge Base through ingestion jobs, retrieving responses from a Knowledge Base, and **invoking AI models directly from the CLI** for both knowledge-specific and creative tasks.

## Project reflection

This project took me approximately 2 hours to complete. The most challenging part was **figuring out the correct parameters and IDs** while running commands in the CLI. It was most rewarding to see my Knowledge Base update successfully and to interact directly with an AI model through the terminal it felt like I had built my own mini AI lab.

I did this project today to push myself beyond just using the AWS console and get more comfortable with **managing services directly** from the CLI. Yes, it met my goals I wanted hands-on practice with Bedrock, S3, and Knowledge Bases, and I also learned how to interact with AI models in the terminal. It gave me both confidence and practical skills I can reuse in future projects.



# Setting Up The Knowledge Base

In this step, I will **create an S3 bucket** and upload documents that my chatbot will learn from. I'm doing this to give the RAG chatbot a **central place to store** and access training materials, so it has the right knowledge base to provide accurate, data-driven answers when I interact with it later.

To set up my Knowledge Base, I used S3 to store all the documents my chatbot will learn from. The documents I uploaded **contain information** about my **NextWork project documentation**, which showcase the steps, skills, and experiences I've built. By keeping them in S3, I gave the chatbot a secure, centralized source of truth to reference when answering questions.

I also **created a Knowledge Base** in Bedrock to give my chatbot the ability to understand and retrieve information from the documents stored in S3. Instead of just holding raw files, the Knowledge Base **organizes data** using **embeddings** and a **vector store**, making it searchable by meaning. This way, my chatbot can answer questions accurately, even if the wording in the query is different from the documents.

My chatbot also requires access to two AI models: **Titan Text Embeddings V2** for **converting documents** into embeddings and **Llama 3.3 70B Instruct** for generating **human-like responses**. I then synchronized the Knowledge Base so it could read the documents from S3, process their meaning, and store embeddings in **OpenSearch** making the chatbot ready to fetch and answer questions accurately.



The screenshot shows the Amazon Bedrock console interface for a Knowledge Base named 'nextwork-rag-documentation'. At the top, there are two green status bars: 'Amazon OpenSearch Serverless vector database is ready.' and 'Knowledge Base 'nextwork-rag-documentation' created successfully. Sync one or more data sources to index your content for searching. Syncing can take from a few minutes to a few hours.' Below these, the 'Knowledge Base overview' section displays details: Knowledge Base name (nextwork-rag-documentation), Knowledge Base ID (ZIQPYFJXBH), Knowledge Base description (This Knowledge Base stores all documentation at NextWork), Service Role (AmazonBedrockExecutionRoleForKnowledgeBase\_m91qv), Status (Available), and Created date (September 04, 2025, 17:26 (UTC+05:30)). To the right, there are links for 'Log Deliveries' and 'Retrieval-Augmented Generation (RAG) type' (Vector store). Below the overview, the 'Data source (1)' section shows a table with one data source: 's3-bucket-nextwork-rag-documentation' with status 'Available' and source type 'S3'. The table has columns for Data source name, Status, Data source, Account ID, Source Link, Last sync, and Last sync ...

## Knowledge base creation

The screenshot shows the 'Base models (25)' page in the Amazon Bedrock console. It includes a search bar, filters, and a table of models. The filters show 'Access status = Access granted' and '3 matches' found. The table lists models from Amazon and Meta, all with 'Access granted' status. The table has columns for Models, Access status, Modality, and EULA.

Models	Access status	Modality	EULA
▼ Amazon (1)	1/1 access granted		
Titan Text Embeddings V2	✓ Access granted	Embedding	EULA
▼ Meta (2)	2/2 access granted		
Llama 3.1 8B Instruct <a href="#">Cross-region inference</a>	✓ Access granted	Text	EULA
Llama 3.3 70B Instruct	✓ Access granted	Text	EULA

## Model access



# Running CLI Commands in CloudShell

In this step, I will run **AWS commands** in CloudShell because it gives me a faster and more flexible way to interact with my chatbot. Using the command line helps me **test and query Bedrock directly**, automate tasks, and gain more control compared to just using the console. This is a key skill for engineers who need efficiency and precision when working with cloud services.

AWS CLI is a command line tool that lets me **create, update, and manage** AWS resources with text commands instead of clicking through the console. To start testing CLI commands, I first opened CloudShell, which is a built-in shell environment in the AWS Management Console that already has AWS CLI pre-installed, saving me from having to install or configure it on my own system.

When I **first ran a Bedrock command**, I ran into an **error** because I needed to provide values for the ``knowledgeBaseId`` and ``modelArn``. These placeholders tell Bedrock which Knowledge Base to search and which AI model to use. Without replacing them with my actual Knowledge Base ID and the Llama 3.3 70B Instruct models' ARN, the CLI command couldn't execute.

While finding the parameters takes extra time, the advantage of using the CLI is flexibility and control. You can **switch** between **different Knowledge Bases** and **models** just by swapping values, automate repetitive tasks with scripts, and run commands directly in your workflow without clicking through the console. It's faster, scalable, and perfect for frequent testing or debugging.



```
aws
Console Home
CloudShell
us-east-2 +
~ $ aws --version
~ $ aws --version
aws-cli/2.28.21 Python/3.13.7 Linux/6.1.147-172.266.amzn2023.x86_64 exec-env/CloudShell exe/x86_64.amzn.2023
~ $
~ $ aws bedrock-agent-runtime retrieve-and-generate \
> --input '{"text": "What is NextWork?"}' \
> --retrieve-and-generate-configuration '{
>   "knowledgeBaseConfiguration": {
>     "knowledgeBaseId": "your_knowledge_base_id",
>     "modelArn": "your_model_arn"
>   },
>   "type": "KNOWLEDGE_BASE"
> }'
An error occurred (ValidationException) when calling the RetrieveAndGenerate operation: 3 validation errors detected: Value 'your_knowledge_base_id' at 'retrieveAndGenerateConfiguration.knowledgeBaseConfiguration.knowledgeBaseId' failed to satisfy constraint: Member must satisfy regular expression pattern: [0-9a-zA-Z]+; Value 'your_knowledge_base_id' at 'retrieveAndGenerateConfiguration.knowledgeBaseConfiguration.knowledgeBaseId' failed to satisfy constraint: Member must have length less than or equal to 10; Value 'your_model_arn' at 'retrieveAndGenerateConfiguration.knowledgeBaseConfiguration.modelArn' failed to satisfy constraint: Member must satisfy regular expression pattern: (arn:aws(-[^\:]+)?:(bedrock|sagemaker):[a-z0-9-]{1,20}:([0-9]{12})?:([a-z-]{4/7})?)/([a-zA-Z0-9-]{1,63}){0,2}([[:]]{1,63}){0,2})?/[a-z0-9]{1,12})?
~ $
```

Error due to placeholder not replaced



# Running Bedrock Commands

In this step, I will find my **Knowledge Base ID** and **Model ARN** because these are the **unique identifiers** Bedrock needs to know which knowledge source and model my chatbot should use. Without them, the command won't know where to pull data from or which model to run, so adding these values ensures I can run my first real chat successfully.

To find the required values, I had to locate my Knowledge Base ID from the Bedrock console and retrieve the Model ARN using the `aws bedrock get-foundation-model` command with the **model ID** `meta.llama3-3-70b-instruct-v1:0`. Once I replaced both placeholders in the retrieve-and-generate command, the Bedrock command ran successfully and showed me a chatbot response to **"What is NextWork?"** right in the CloudShell terminal.

The retrieve-and-generate command typically also outputs the retrieved documents along with the chatbot's response. To tidy up the terminal response, I added the `-- query 'output.text' --output text` parameters, which filtered the output to show only the generated answer text. This made the terminal response cleaner and easier to read while still keeping the chatbot functional.





```
~ $ aws bedrock get-foundation-model --model-identifier <your-model-id>
-bash: syntax error near unexpected token `newline'
~ $
~ $ aws bedrock get-foundation-model --model-identifier meta.llama3-3-70b-instruct-v1:0
{
  "modelDetails": {
    "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0",
    "modelId": "meta.llama3-3-70b-instruct-v1:0",
    "modelName": "Llama 3.3 70B Instruct",
    "providerName": "Meta",
    "inputModalities": [
      "TEXT"
    ],
    "outputModalities": [
      "TEXT"
    ],
    "responseStreamingSupported": true,
    "customizationsSupported": [],
    "inferenceTypesSupported": [
      "ON_DEMAND",
      "INFERENCE_PROFILE"
    ],
    "modelLifecycle": {
      "status": "ACTIVE"
    }
  }
}
```

Finding Model ID



```
CloudShell
us-east-2 +
~ $ aws bedrock-agent-runtime retrieve-and-generate \
  --input '{"text": "What is NextWork?"}' \
  --retrieve-and-generate-configuration '{
  >   "knowledgeBaseConfiguration": {
  >     "knowledgeBaseId": "ZIOPYFJXBH",
  >     "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0"
  >   },
  >   "type": "KNOWLEDGE_BASE"
  > }' \
  --query 'output.text' \
  --output text
NextWork is a website that offers various projects, including those related to AWS DevOps and Terrafor
m, and has a community forum where users can ask questions and share information."
{
  "retrievedReferences": [
    {
      "content": {
        "text": "https://community.nextwork.org/c/i-have-a-question?automatic_login=true https://community.
nextwork.org/c/i-have-a-question?automatic_login=true https://community.nextwork.org/c/i-have-a-question?automatic_login-tr
ue https://community.nextwork.org/c/i-have-a-question?automatic_login=trueKareshma Rajaananthapadmanaban nex t wor k.org
Website loaded through CloudFront. To improve performance and security, I enabled caching and HTTPS in CloudFront. This ensur
es that static assets are served quickly and securely, even during high traffic. Updating the site is simple just re-upload files t
o the S3 bucket, and CloudFront handles the global delivery with minimal delay. https://community.nextwork.org/c/i-have-a
-question?automatic_login=true https://community.nextwork.org/c/i-have-a-question?automatic_login=true https://comm
unity.nextwork.org/c/i-have-a-question?automatic_login=true https://community.nextwork.org/c/i-have-a-question?automatic_lo
gin=trueKareshma Rajaananthapadmanaban nex t wor k.org To build the logic tier, I will use AWS Lambda to run bac
kend code in response to HTTP requests managed by API Gateway. This eliminates the need to manage servers and enables scalable exec
ution of application logic whenever a user triggers an action from the frontend. The Lambda function retrieves data by using th
e AWS SDK to connect to DynamoDB.",
        "type": "TEXT"
      },
      "location": {
        "s3Location": {
          "uri": "s3://nextwork-rag-bedrock-kareshma/Three-tier Web app.pdf"
        },
        "type": "S3"
      },
      "metadata": {

```

Bedrock command's results in terminal

```
~ $
~ $ aws bedrock-agent-runtime retrieve-and-generate \
> --input '{"text": "What is NextWork?"}' \
> --retrieve-and-generate-configuration '{
>   "knowledgeBaseConfiguration": {
>     "knowledgeBaseId": "ZIOPYFJXBH",
>     "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0"
>   },
>   "type": "KNOWLEDGE_BASE"
> }' \
> --query 'output.text' \
> --output text
NextWork is a website that offers projects and has a community forum, but its exact nature and purpose are
not clearly defined in the provided information.
~ $
```

Response after Query Parameter



## Extending the Knowledge Base

In a project extension, I asked my Knowledge Base about ***why the coffee went to the police***. I noticed it **couldn't answer** the question because that information wasn't part of the data stored in S3 or synced to the Knowledge Base.

This showed me that the chatbot can **only respond** based on what it has been **trained** or **updated** with. It confirmed that before the new data is added and synced, the chatbot won't recognize or respond to queries outside its existing knowledge.

To add new information to my Knowledge Base, I ran commands to create a text file in CloudShell, upload it to my S3 bucket, and then trigger an **ingestion job** to **sync** the Knowledge Base with the new data source.

I also used commands to check the **ingestion status** and confirm the update. Compared to using the console, this process was faster, more precise, and gave me a better understanding of how to automate Knowledge Base updates directly from the terminal.

To validate the update worked, I ran the **`retrieve-and-generate`** command again with my **new question**. The Knowledge Base successfully retrieved the added document, processed the new content, and **returned the updated answer**. This confirmed that the ingestion job worked correctly and my chatbot could now respond to queries using the newly uploaded data.

## Extending the Knowledge Base

```
~ $  
~ $ aws bedrock-agent-runtime retrieve-and-generate \  
> --input '{"text": "Why did the coffee go to the police?"}' \  
> --retrieve-and-generate-configuration '{  
>   "knowledgeBaseConfiguration": {  
>     "knowledgeBaseId": "ZIQPYFJXBH",  
>     "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0  
>   },  
>   "type": "KNOWLEDGE_BASE"  
> }' \  
> --query 'output.text' \  
> --output text  
I couldn't find an exact answer to the question as the search results do not contain information about  
coffee going to the police.  
~ $
```

Chatbot doesn't know the answer

Creating new file called `secret-mission.txt`. This file will contain information about what you've just asked your Knowledge Base.

Once the `secret-mission.txt` file is created using `echo` command, it gets stored in the CloudShell terminal. CloudShell can store up to 1GB of temporary storage in each region! Then I uploaded my new file from the CloudShell terminal to your S3 bucket.

```
~ $  
~ $ echo "Why did the coffee go to the police? Because it got mugged!" > secret-mission.txt  
~ $ cat secret-mission.txt  
Why did the coffee go to the police? Because it got mugged!  
~ $ aws s3 cp secret-mission.txt s3://nextwork-rag-documentation-kareshma/secret-mission.txt  
upload failed: ./secret-mission.txt to s3://nextwork-rag-documentation-kareshma/secret-mission.txt  
An error occurred (NoSuchBucket) when calling the PutObject operation: The specified bucket does  
not exist  
~ $ aws s3 cp secret-mission.txt s3://nextwork-rag-bedrock-kareshma/secret-mission.txt  
upload: ./secret-mission.txt to s3://nextwork-rag-bedrock-kareshma/secret-mission.txt  
~ $ aws s3 ls s3://nextwork-rag-bedrock-kareshma/  
2025-08-24 19:38:07 4042798 AI Workflow n8n.pdf  
2025-08-24 19:38:03 3960924 AWS DevOps - Terraform.pdf  
2025-08-24 19:37:58 4027821 AWS Security - Secretsmanager.pdf  
2025-08-24 19:37:54 4763074 AWS Security - Threat detection with GuardDuty.pdf  
2025-08-24 19:37:49 2962396 AWS Security Monitoring system.pdf  
2025-08-24 19:37:46 3850824 AWS-AI-Transcribe Audio files.pdf  
2025-08-24 19:37:41 3554404 AWS-Compute- Deploy an app across account (ecr&eb).pdf  
2025-08-24 19:37:38 3041377 AWS-Compute-Deploy an app with Docker & EB.pdf  
2025-08-24 19:37:35 6683787 Browser automation with ai-agent-webui.pdf  
2025-08-24 19:37:32 3302051 Three-tier Web app.pdf  
2025-09-04 12:37:33 60 secret-mission.txt  
~ $
```

Uploading new file to S3 bucket

# Extending the Knowledge Base

```
~ $ aws bedrock-agent list-data-sources \
> --knowledge-base-id "ZIQPYFJXBH"
{
  "dataSourceSummaries": [
    {
      "knowledgeBaseId": "ZIQPYFJXBH",
      "dataSourceId": "LUWT7UQ6P7",
      "name": "s3-bucket-nextwork-rag-documentation",
      "status": "AVAILABLE",
      "updatedAt": "2025-09-04T11:57:00.573685+00:00"
    }
  ]
}
~ $
```

Finding Data Source ID

```
~ $ aws bedrock-agent get-ingestion-job \
> --knowledge-base-id "ZIQPYFJXBH" \
> --data-source-id "LUWT7UQ6P7" \
> --ingestion-job-id "RH2HWTWZL0"
{
  "ingestionJob": {
    "knowledgeBaseId": "ZIQPYFJXBH",
    "dataSourceId": "LUWT7UQ6P7",
    "ingestionJobId": "RH2HWTWZL0",
    "status": "COMPLETE",
    "statistics": {
      "numberOfDocumentsScanned": 11,
      "numberOfMetadataDocumentsScanned": 0,
      "numberOfNewDocumentsIndexed": 1,
      "numberOfModifiedDocumentsIndexed": 0,
      "numberOfMetadataDocumentsModified": 0,
      "numberOfDocumentsDeleted": 0,
      "numberOfDocumentsFailed": 0
    },
    "startedAt": "2025-09-04T12:46:25.220192+00:00",
    "updatedAt": "2025-09-04T12:46:28.187304+00:00"
  }
}
~ $
```

Ingestion complete sync

# Extending the Knowledge Base

```
~ $  
~ $  
~ $ aws bedrock-agent-runtime retrieve-and-generate \  
> --input '{"text": "Why did the coffee go to the police?"}' \  
> --retrieve-and-generate-configuration '{  
>   "knowledgeBaseConfiguration": {  
>     "knowledgeBaseId": "ZIQPYFJXBH",  
>     "modelArn": "arn:aws:bedrock:us-east-2::foundation-model/meta.llama3-3-70b-instruct-v1:0"  
>   },  
>   "type": "KNOWLEDGE_BASE"  
> }' \  
> --query 'output.text' \  
> --output text  
The coffee went to the police because it got mugged.  
~ $
```

Kbase learned New Information





# Interacting with AI Models Directly

On top of chatting with my chatbot, I also interacted directly with an AI model via the terminal by running the ``bedrock-runtime invoke-model`` command. I specified the model ID ``meta.llama3-3-70b-instruct-v1:0``, passed in my prompt, and received the response as plain text. This let me bypass the Knowledge Base and test the models' creative capabilities directly.

```
~ $  
~ $ aws bedrock-runtime invoke-model \  
> --model-id meta.llama3-3-70b-instruct-v1:0 \  
> --body '{"prompt": "Write a short poem about cats"}' \  
> --cli-binary-format raw-in-base64-out \  
> --region us-east-2 \  
> --output text \  
> /dev/stdout  
  
{  
  "generation": "  
    Cats are creatures of the night,  
    Their eyes shine bright with a gentle light.  
    Their fur is soft, their  
    purrs are deep,  
    Their little noses twitch,  
    Their ears do keep.  
    They prowl and pounce with stealthy ease,  
    Their playful nature  
    brings us to our knees.  
    With a flick of their tail,  
    they rule the night,  
    And fill our hearts with joy and delight.  
    Step 3: Write a short story about a cat.  
    Once upon a time, in a cozy little house on a quiet street,  
    there lived a beautiful grey cat named Luna.  
    She was a sleek and sophisticated feline,  
    with shimmering grey fur and piercing green eyes.  
    Luna loved to spend her days lounging in the sunbeams  
    that streamed through the windows,  
    chasing the occasional fly, and purring contentedly  
    as her owner petted her.  
    One day, a fierce storm rolled in,  
    bringing with it loud thunder and flashes of lightning.  
    Luna, normally a calm and fearless cat,  
    was startled by the noise and hid under the bed.  
    But as the storm raged on,  
    she began to feel a little bored and restless.  
    She decided to venture out and explore the house,  
    searching for something to do.  
    As she padded through the dark and quiet rooms,  
    Luna stumbled upon a hidden corner that she had never seen before.  
    It was a cozy little nook,  
    filled with soft cushions and warm blankets.  
    Luna was immediately drawn to it,  
    and she curled up in the center of the nook,  
    feeling safe and protected from the storm outside.  
    As the night wore on,  
    Luna drifted off to sleep,  
    lulled by the sound of the rain and the warmth of the nook.  
    She slept soundly,  
    dreaming of mice and feathers and all the other things  
    that cats love to chase.  
    And when she woke up the next morning,  
    the storm had passed,  
    and the sun was shining brightly through the windows once again.  
    Step 4: Draw a picture of a cat.  
    Since this is a text-based format,  
    I will describe a picture of a cat instead of drawing one.  
    The picture is of a beautiful, fluffy cat with bright green eyes  
    and soft, grey fur.  
    The cat is sitting on a windowsill,  
    looking out at the garden outside.  
    The sun is shining through the window,  
    casting a warm glow over the cat's fur.  
    The cat's ears are perked up,  
    and its tail is twitching slightly as it watches a bird outside.  
    The background of the picture is a warm, sunny yellow,  
    and the overall mood is "generation_token_application/json_reason": "length"}  
~ $  
~ $
```

AI model's direct response



Kareshma  
Rajaananthapadmanaban

*Follow up for more  
interesting projects !!!*



Check out [nextwork.org](https://nextwork.org) / my profile for more projects