Serverless Named Entity Recognition on AWS Lambda

Riccardo Pitzanti - 1947877 pitzanti.1947877@studenti.uniroma1.it
Federico Iannini - 1931748 iannini.1931748@studenti.uniroma1.it
Sapienza Università di Roma

August 21, 2025

1 Introduction

2 Background

2.1 Named Entity Recognition (NER) and spaCy

Named Entity Recognition (NER) is a natural language processing (NLP) task focused on identifying and categorizing information elements, known as entities, within unstructured text. These entities are typically classified into predefined categories such as persons (PERSON), organizations (ORG), and geopolitical entities (GPE). The spaCy library in Python provides a production-ready framework for implementing NLP pipelines, offering statistical models for tasks like NER.

2.2 AWS SAM & Containerized Build with Docker

The AWS Serverless Application Model (SAM) is an open-source framework that extends AWS CloudFormation to provide a simplified syntax for defining serverless resources. It is a form of Infrastructure as Code (IaC) specifically designed to express the functions, APIs, permissions, and events that compose a serverless application. A containerized build process involves using a consistent, isolated environment to compile application dependencies and package the code. By employing Docker, SAM ensures that the build process is executed within a containerized environment that replicates the AWS Lambda runtime. This guarantees consistency between the development build and the deployment target, thereby mitigating compatibility issues.

2.3 AWS Lambda and HTTP API

AWS Lambda is a serverless, event-driven compute service that allows for the execution of code in response to triggers without requiring the management of underlying servers. It automatically scales with incoming request volume and utilizes a fine-grained cost model based on actual compute consumption. The Amazon API Gateway is a fully managed service that simplifies the creation, publication, and maintenance of secure APIs at any scale. It acts as a front-door for applications to access data and business logic from backend services, providing essential features like HTTP endpoint management, request routing, and authorization.

2.4 Observability with CloudWatch

Observability is a system property that describes how well internal states can be understood from external outputs, primarily through the collection and analysis of logs, metrics, and traces. Amazon CloudWatch is a monitoring and observability service that provides a unified view of AWS resource health, application performance, and operational trends. It automatically collects performance metrics from services like AWS Lambda.

2.5 Load Testing with Locust

Locust is an open-source load testing tool that allows developers to define user behavior with Python code and simulate various concurrent users to assess a system's performance under stress. Its distributed and scalable nature makes it suitable for testing the limits of web services and APIs. It provides a real-time web-based user interface to visualize performance indicators such as requests per second, response times, and the number of failing requests as the test is running.

3 System Design and Implementation

3.1 Architecture overview

- **Client** sends `POST /ner` with a short text body.
- **API Gateway (HTTP API)** forwards requests to Lambda.
- **Lambda** parses input, runs spaCy NER (warm model), and returns spans.
- **CloudWatch** records metrics and logs for observability.

API contract.

```
POST /ner
Content-Type: application/json
{"text": "Alan Turing was born on June 23, 1912, in London, England."}

200 OK
{"entities": [
    {'text': 'Alan Turing', 'label': 'PERSON', 'start': 0, 'end': 11},
    {'text': 'June 23, 1912', 'label': 'DATE', 'start': 24, 'end': 37},
    {'text': 'London', 'label': 'GPE', 'start': 41, 'end': 47},
    {'text': 'England', 'label': 'GPE', 'start': 49, 'end': 56}]
}
```

4 Methodology and Implementation

4.1 Inference Core Module (`src/ner.py`)

This module constitutes the computational core responsible for the Named Entity Recognition (NER) task. It utilizes the spaCy library, a framework for natural language processing (NLP) in Python. The pre-trained statistical model (`en_core_web_sm`) is loaded into a global constant at module import time. The primary function, `extract_entities(text: str)`, processes an input string through the spaCy pipeline. It returns a list of dictionaries, each containing the extracted entity's surface form (`text`), its ontological class (`label`), and the character-level indices (`start`, `end`) denoting its span within the original text.

4.2 Lambda Handler Function (`src/handler.py`)

This module implements the AWS Lambda function handler, which serves as the entry point for requests proxied by Amazon API Gateway. Its primary role is to manage the HTTP request-response cycle. The handler first invokes a helper function, `_parse_body`, to normalize the incoming event structure. This function abstracts away the differences between the event payload delivered by API Gateway (where the HTTP request body is passed as a JSON-encoded string)

and the event object used during local testing (which may be a Python dictionary). The handler then performs input validation, checking for the presence and type of the required `'text'` field. Invalid requests result in a `400 Bad Request` response. Validated text is passed to the inference core, and the resulting entities are serialized into a JSON object returned within a `200 OK` response, complete with appropriate HTTP headers for content type.

4.3 Build Process

The build is executed using the AWS SAM CLI command `sam build -use-container`. This process constructs the deployment package inside a Docker container that emulates the Amazon Linux environment of AWS Lambda.

4.4 Infrastructure Provisioning (SAM Template)

The cloud infrastructure is defined declaratively using the AWS Serverless Application Model (SAM), an extension of AWS CloudFormation. The template, `template.yaml`, specifies a minimal and functional stack:

- A single AWS Lambda function resource with its runtime (`python3.9`), allocated memory (512 MB), and timeout (15 seconds) defined in the `Globals` section.
- An Amazon API Gateway HTTP API resource, which provisions a managed HTTPS endpoint. This API is configured with a single route (`POST /ner`) that integrates directly with the Lambda function.
- A pre-existing IAM role (`LabRole`) is referenced for execution permissions, a constraint of the AWS Academy Learner Lab environment. In a standard AWS account, SAM would typically generate a minimal role with necessary permissions automatically.

This Infrastructure-as-Code (IaC) approach guarantees that the entire application stack is versioned, reproducible, and deployable with a single command.

4.5 Front-End

5 Performance Evaluation and Testing

6 Results

7 Limitations and Future Work

8 Conclusion

Reproducibility & Repository Notes

The repository is organized for "clone → run" on Windows:

1. Install Python 3.10+, Docker Desktop, AWS CLI v2, AWS SAM CLI.
2. `python -m venv .venv; .  .\.venv\Scripts\Activate.ps1`
3. `pip install -r requirements.txt`
4. `sam build -use-container`
5. `sam deploy -guided -profile learnerlab`

6. Use `scripts/smoke.ps1` to POST `/ner`.

7. Tear down with `scripts/delete.ps1`.

Ethical, Cost, and Budget Considerations

We designed for negligible standing cost and minimal environmental impact: no always-on compute, small artifacts, and conservative logs. Load tests remain moderate to avoid unintended denial-of-service behavior and unnecessary spend.

References

[1] ExplosionAI. *spaCy*. https://spacy.io/.

[2] ExplosionAI. *NER in spaCy*. https://spacy.io/api/entityrecognizer.

[3] AWS. *AWS Serverless Application Model (SAM)*. https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/.

[4] AWS. *AWS Lambda Developer Guide*. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html.

[5] AWS. *Amazon API Gateway HTTP APIs*. https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api.html.

[6] AWS. *Amazon CloudWatch Metrics for Lambda*. https://docs.aws.amazon.com/lambda/latest/dg/monitoring-metrics.html.

[7] Locust. *A modern load testing framework*. https://locust.io/.