Backend Infrastructure Documentation

Project Backend Team

February 11, 2026

Contents

1 Overview **3**

2 System Architecture **3**
2.1 Architectural Components . . . 3
2.1.1 Lambda Functions . . . 3
2.1.2 Lambda Layers . . . 4
2.1.3 IAM Permissions . . . 4
2.1.4 AWS CDK Stack . . . 4

3 Project Structure **5**

4 Current Lambda Functions **5**
4.1 PDF Handler . . . 5
4.2 Signup Handler . . . 5
4.3 Login Handler . . . 6

5 CI/CD Pipeline **6**
5.1 Build Stage . . . 6
5.2 Deploy Stage . . . 6

6 Local Testing **7**
6.1 Direct Python Testing . . . 7
6.2 AWS SAM . . . 7

7 Troubleshooting **7**
7.1 Handler Import Error . . . 7
7.2 Missing Dependencies . . . 7
7.3 IAM Access Denied . . . 8

8 Deployment Workflow **8**
8.1 Development Workflow . . . 8
8.2 Production Workflow . . . 8

9 References **8**

1 Overview

This document describes the backend infrastructure for the system, including AWS Lambda functions, Lambda Layers, IAM configuration, CI/CD pipeline setup, and deployment workflow.

The backend is built using:

- Python 3.11
- AWS Lambda
- AWS CDK
- DynamoDB
- GitLab CI/CD

Common issues typically involve:

- Missing Python dependencies
- Incorrect Lambda Layer configuration
- Insufficient IAM permissions
- AWS credential misconfiguration

2 System Architecture

2.1 Architectural Components

2.1.1 Lambda Functions

Lambda functions implement individual backend operations such as:

- PDF processing
- User registration
- User authentication

Each handler:

- Receives an `event` and `context`
- Returns a dictionary containing `statusCode` and `body`

Example handler structure:

```
def handler_name(event, context):
    try:
        return {
            "statusCode": 200,
            "body": json.dumps({"message": "Success"})
        }
    except Exception as e:
        return {
            "statusCode": 400,
            "body": json.dumps({"error": str(e)})
        }
```

2.1.2 Lambda Layers

Lambda Layers package reusable dependencies that can be shared across multiple Lambda functions.

Examples:

- PDF dependencies (PyPDF2)
- Registration dependencies (boto3)

Layers reduce duplication and minimize deployment package size.

2.1.3 IAM Permissions

IAM policies define which AWS services each Lambda function may access.

If a Lambda function fails with an `AccessDeniedException`, verify that the required permissions are added in `backend_stack.py`:

```
self.add_permissions(
    my_lambda,
    ["dynamodb:GetItem", "dynamodb:PutItem"],
    ["arn:aws:dynamodb:*:*:table/Users"]
)
```

2.1.4 AWS CDK Stack

The CDK stack defines:

- Lambda functions
- Lambda layers
- IAM roles and permissions
- DynamoDB resources

Primary stack file:

```
backend/backend_stack.py
```

3 Project Structure

```
Backend/
          README.md
          app.py
          cdk.json
          requirements.txt
          backend/
                backend_stack.py
          layers/
                pdf_dependencies/
                registration_dependencies/
          src/
              handlers/
                  pdf_handler.py
                  signup_handler.py
                  login_handler.py
```

4 Current Lambda Functions

4.1 PDF Handler

File: `pdf_handler.py`

- Extracts text from PDF documents
- Uses PyPDF2
- Requires pdf_dependencies layer

4.2 Signup Handler

File: `signup_handler.py`

- Creates new user accounts
- Stores credentials in DynamoDB Users table
- Requires boto3 via registration_dependencies layer

4.3 Login Handler

File: `login_handler.py`

- Authenticates users
- Validates email and password
- Reads from DynamoDB Users table

5 CI/CD Pipeline

5.1 Build Stage

The GitLab pipeline performs the following:

1. Builds Lambda Layer zip files
2. Packages each Lambda handler individually
3. Stores artifacts for deployment

Example layer build step:

```
cd Backend/layers/my_dependencies
pip install -r requirements.txt -t python/
zip -r ../../my_dependencies_layer.zip python
```

Example handler packaging:

```
cd Backend/src/handlers
zip ../../../my_handler.zip my_handler.py
```

5.2 Deploy Stage

Deployment includes:

- Running `cdk bootstrap`
- Executing `cdk deploy`
- Branch-based environment selection:
  - main → production
  - other branches → development

6 Local Testing

6.1 Direct Python Testing

```
from src.handlers.my_handler import my_handler

event = {"body": "test"}
response = my_handler(event, None)
print(response)
```

6.2 AWS SAM

```
sam local start-api
```

7 Troubleshooting

7.1 Handler Import Error

Error:

```
Unable to import module
```

Solution:

- Verify handler filename
- Confirm correct handler reference in CDK
- Ensure zip structure is correct

7.2 Missing Dependencies

Error:

```
ModuleNotFoundError
```

Solution:

- Add dependency to requirements.txt
- Rebuild layer
- Reattach layer in CDK

7.3 IAM Access Denied

Solution:

- Add required service actions
- Verify correct resource ARN
- Redeploy stack

8 Deployment Workflow

8.1 Development Workflow

1. Create feature branch
2. Implement changes
3. Push to repository
4. Trigger CI/CD pipeline
5. Verify in development environment

8.2 Production Workflow

1. Merge into main branch
2. CI/CD triggers production deployment
3. Verify production environment

9 References

- AWS CDK Documentation: https://docs.aws.amazon.com/cdk/
- AWS Lambda Layers: https://docs.aws.amazon.com/lambda/latest/dg/packaging-layers.html
- AWS Lambda Best Practices: https://docs.aws.amazon.com/lambda/latest/dg/lambda-best-practices.html
- GitLab CI/CD Documentation: https://docs.gitlab.com/ee/ci/