ChatGPT

AWS CLI Deployment Guide (Next.js Frontend + Python Serverless Backend)

Prerequisites: Install and configure the AWS CLI with your credentials (see AWS documentation) before proceeding. This guide uses official AWS docs and trusted tutorials for each step.

Frontend: Next.js Static Site on S3 + CloudFront

Deploy the React/Next.js + Tailwind frontend as a static site. After building and exporting your Next.js app (e.g. `npm run build && npm run export` to produce an `out/` directory), create an S3 bucket for hosting. For example:

- **Create S3 bucket:**

  ```
  aws s3 mb s3://my-nextjs-app --region us-east-1
  ```

  (This command creates a new bucket named `my-nextjs-app`.) [1] .

- **Enable static website hosting (optional):** (You can use `aws s3api put-bucket-website` with an `IndexDocument`.) For simplicity, you can host via CloudFront as the HTTPS endpoint anyway.

- **Upload static files:** Sync the local export directory to S3:

  ```
  aws s3 sync ./out s3://my-nextjs-app
  ```

  This copies all static files into the bucket [2] . If you enable website hosting directly, set the bucket's index document (e.g. `index.html`).

- **Create CloudFront distribution:** To serve the site over HTTPS from edge locations, create a CloudFront distribution pointing at the S3 bucket origin. Using the AWS CLI, you typically supply a JSON config file; for example:

```
aws cloudfront create-distribution --distribution-config file://
distribution.json
```

This command (illustrated in the AWS CLI examples) creates a CDN for your S3 site [3] . The output will include a `DomainName` like `dXXXXX.cloudfront.net`. You can later invalidate the cache with:

```
aws cloudfront create-invalidation --distribution-id <DIST_ID> --paths "/*"
```

See the AWS CLI CloudFront guide for details on the distribution JSON format [3]. By default, CloudFront provides an HTTPS endpoint (using the `*.cloudfront.net` domain). If you have a custom domain, configure an ACM SSL certificate and set `Aliases` in the distribution config.

Backend: AWS Lambda (Python) + API Gateway

Build your backend as Python Lambda functions exposing APIs via API Gateway:

- **Create IAM role for Lambda:** Before creating functions, create an execution role with necessary permissions (e.g. allowing Lambda to write to DynamoDB, invoke other AWS services, etc.). You can use `aws iam create-role` and `aws iam put-role-policy`, but specifics depend on your app needs.

- **Create Lambda functions:** Zip each Python function and use AWS CLI to create it. For example, to create a function named `SubscribeHandler` with runtime Python 3.9:

  ```
  aws lambda create-function \
      --function-name SubscribeHandler \
      --runtime python3.9 \
      --role arn:aws:iam::123456789012:role/lambda-execution-role \
      --handler subscribe.handler \
      --zip-file fileb://subscribe.zip
  ```

  (This matches the AWS example pattern shown in the docs [4], but using Python runtime and your handler names.) After creating, note the function ARN for API Gateway integration.

- **Set up API Gateway (REST API):** Use AWS CLI to define a REST API and link it to the Lambda functions. For each function, you'll create an API, resources, methods, and integrations. For example (from AWS docs):

- **Create the REST API:**

  ```
  aws apigateway create-rest-api --name 'MyAPI'
  ```

  Capture the returned `id` (the API identifier) [5].

- **Create resources and methods:** Suppose you want a `/subscribe` resource and allow ANY method (proxy). First, get the root resource ID (from the create-rest-api output) and then:

```
aws apigateway create-resource \
    --rest-api-id <api-id> \
    --parent-id <root-id> \
    --path-part subscribe
aws apigateway put-method \
    --rest-api-id <api-id> \
    --resource-id <resource-id> \
    --http-method ANY \
    --authorization-type NONE
```

These commands create a `/subscribe` path and enable an `ANY` method on it (AWS CLI example for proxy integrations uses ANY on a `{proxy+}` path) [6].

- **Integrate with Lambda:** Attach the API method to the Lambda function using a proxy integration:

```
aws apigateway put-integration \
    --rest-api-id <api-id> \
    --resource-id <resource-id> \
    --http-method ANY \
    --type AWS_PROXY \
    --integration-http-method POST \
    --uri arn:aws:apigateway:us-east-1:lambda:path/2015-03-31/functions/
<lambda-function-arn>/invocations \
    --credentials arn:aws:iam::123456789012:role/apigAwsProxyRole
```

This tells API Gateway to invoke your Lambda function for all requests to `/subscribe` [7]. (You must replace `<lambda-function-arn>` and adjust region/account. The `--credentials` role must allow API Gateway to invoke the function; alternatively you can use `aws lambda add-permission` to allow API Gateway as principal.)

- **Deploy the API:** Finally, deploy your API to a stage (e.g. “prod”):

```
aws apigateway create-deployment --rest-api-id <api-id> --stage-name prod
```

The AWS CLI docs show that as the last step to make your endpoints live [8].

After deployment, API Gateway returns an invoke URL (like `https://<api-id>.execute-api.us-east-1.amazonaws.com/prod`). Test it with `curl` or a browser.

Data Storage: Amazon DynamoDB (Subscriptions/Feedback)

Use DynamoDB tables for storing subscriptions and feedback data. With AWS CLI:

- **Create DynamoDB tables:** For each use case, run `aws dynamodb create-table`. For example, to create a `Subscriptions` table with primary key `email` (string):

  ```
  aws dynamodb create-table \
      --table-name Subscriptions \
      --attribute-definitions AttributeName=email,AttributeType=S \
      --key-schema AttributeName=email,KeyType=HASH \
      --billing-mode PAY_PER_REQUEST
  ```

  (This follows the CLI example pattern: define attributes and key schema for the table [9].) Do likewise for a `Feedback` table (e.g. with keys `id` or `timestamp`). The command returns a JSON with table details. Ensure the table status becomes `ACTIVE` before use.

Your Lambda functions (via AWS SDK or boto3) can then use those table names. You only need CLI to create the tables initially (optionally, insert test items with `aws dynamodb put-item`).

Scheduled Processing: EventBridge (Weekly Summary)

Set up a CloudWatch Events/EventBridge rule to trigger a weekly summary Lambda:

- **Create schedule rule:** Use `aws events put-rule` with a cron or rate expression. For example, to run every week:

  ```
  aws events put-rule \
      --name WeeklySummaryRule \
      --schedule-expression 'rate(7 days)'
  ```

  This creates a scheduled rule that emits an event every 7 days [10].

- **Add Lambda target:** Grant permission for the rule to invoke your summary Lambda, then add the Lambda as a target. In AWS CLI, first allow EventBridge to invoke the function:

  ```
  aws lambda add-permission \
      --function-name WeeklySummaryFunction \
      --statement-id weekly-summary-evt \
      --action 'lambda:InvokeFunction' \
      --principal events.amazonaws.com
  ```

  Then attach the rule's target:

```
aws events put-targets \
    --rule WeeklySummaryRule \
    --targets Id=1,Arn=arn:aws:lambda:us-
east-1:123456789012:function:WeeklySummaryFunction
```

(These steps follow the AWS scheduled Lambda tutorial pattern [11].) Now the Lambda will be invoked once a week by EventBridge.

Email Notifications: Amazon SES

Send emails (e.g. newsletters or summaries) via AWS SES using the CLI:

- **Verify an identity:** Before sending, verify your "From" email or domain. For example:

```
aws ses verify-email-identity --email-address user@example.com
```

This sends a verification email to `user@example.com` that you must confirm [12]. Alternatively, verify a domain with `aws ses verify-domain-identity`.

- **Send email via CLI:** Use `aws ses send-email` or `send-templated-email`. A simple example:

```
aws ses send-email \
    --from sender@example.com \
    --destination ToAddresses=recipient@example.com \
    --message '{"Subject": {"Data": "Hello"}, "Body": {"Text": {"Data":
"Test email"}}}'
```

This uses AWS SES to send a basic email (the AWS CLI docs show a similar `send-email` usage with `--from`, `--destination`, and `--message` files) [13]. For bulk templated emails, see `send-bulk-templated-email` CLI docs. Remember SES is region-specific and often in a sandbox mode for new accounts, so you may need to request production access.

Once identities are verified and limits are satisfied, your Lambda code can invoke SES via AWS SDK (or you can trigger SES send commands via CLI/scripts).

Putting It All Together with AWS CLI

All infrastructure above can be provisioned and managed via AWS CLI commands (no console). Key commands summary:

- **AWS CLI basics:** Ensure AWS CLI v2 is installed and run `aws configure` (to set Access Key, Secret, region, etc.) [14].

- **Frontend hosting:** `aws s3 mb`, `aws s3 sync` [2], `aws cloudfront create-distribution` [3].
- **Lambda functions:** `aws lambda create-function` [4] (plus `update-function-code` on changes).
- **API Gateway:** `aws apigateway create-rest-api` [5], `create-resource`, `put-method`, `put-integration`, `create-deployment` [8].
- **DynamoDB tables:** `aws dynamodb create-table` [9] for each needed table.
- **Scheduled rule:** `aws events put-rule` [10] and related Lambda permission/target commands.
- **Email (SES):** `aws ses verify-email-identity` [12], `aws ses send-email` [13].

By following official tutorials and examples (all cited above), a beginner can replicate each step from VSCode terminal. Each CLI command corresponds to an AWS action in the backend, so you can script out your full deployment (or use shell scripts) to build this full application stack. For more details, consult the linked AWS documentation and guides for each service cited here.

Sources: Official AWS CLI documentation and tutorials for S3, CloudFront, Lambda, API Gateway, DynamoDB, EventBridge, and SES [1] [2] [8] [9] [10] [12] [13].

---

[1] Get started with a secure static website - Amazon CloudFront
https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/getting-started-secure-static-website-cloudformation-template.html

[2] A Guide to Deploy NextJS Application on Amazon S3
https://www.cloudthat.com/resources/blog/a-guide-to-deploy-nextjs-application-on-amazon-s3

[3] [14] AWS CLI & CloudFront: Complete Guide with examples
https://www.learnaws.org/2023/02/02/aws-cli-cloudfront/

[4] Use CreateFunction with an AWS SDK or CLI - AWS Lambda
https://docs.aws.amazon.com/lambda/latest/dg/example_lambda_CreateFunction_section.html

[5] [6] [7] [8] Set up Lambda proxy integration for API Gateway using the AWS CLI - Amazon API Gateway
https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integration-using-cli.html

[9] Step 1: Create a table in DynamoDB - Amazon DynamoDB
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/getting-started-step-1.html

[10] [11] Tutorial: Create an EventBridge scheduled rule for AWS Lambda functions - Amazon EventBridge
https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-run-lambda-schedule.html

[12] [13] Amazon SES examples using AWS CLI - AWS Command Line Interface
https://docs.aws.amazon.com/cli/v1/userguide/cli_ses_code_examples.html