Serverless AWS Journey A Story

Chapter 1: The Idea Sparks

I decided to build a serverless web application no servers, yet full functionality.

Thats when I discovered the AWS Serverless Stack: S3, CloudFront, API Gateway, Lambda, and DynamoDB.

Chapter 2: Building the Frontend Castle (S3 + CloudFront)

- Created an S3 bucket and enabled static website hosting
- Uploaded frontend files using:

  aws s3 sync ./frontend s3://my-app-bucket
- Set up CloudFront to cache and serve content globally

What I learned:

- S3 is perfect for static sites
- CloudFront improves performance and adds security

Chapter 3: Connecting the Backend Bridge (API Gateway)

- Created a REST API with GET and POST methods
- Connected them to Lambda functions
- Enabled CORS for browser compatibility

What I learned:

- API Gateway creates backend APIs with no servers
- CORS is essential for browser-based apps

Chapter 4: Writing the Brain (AWS Lambda)

- Created two Lambda functions:
  - getItemsFunction: fetched data
  - postItemsFunction: saved data
- Used IAM roles for secure access to DynamoDB

What I learned:

- Lambda is powerful and serverless
- IAM roles control permissions
- CloudWatch helps with debugging

Chapter 5: Storing the Knowledge (DynamoDB)

- Created a DynamoDB table with 'id' as primary key
- Used on-demand mode for flexibility

What I learned:

- DynamoDB is a fast NoSQL database
- It integrates seamlessly with Lambda

Chapter 6: Putting It All Together

1. User visits website via CloudFront
2. Static site served from S3
3. Frontend calls API Gateway
4. API Gateway triggers Lambda
5. Lambda accesses DynamoDB
6. Response returned to user

Chapter 7: Things Ill Never Forget

- S3 + CloudFront = static website solution
- API Gateway = serverless API routing
- Lambda = backend logic
- DynamoDB = storage
- IAM = permissions
- CloudWatch = logs & debugging

Bonus Wisdom I Gained

- Serverless is real, powerful, and cost-efficient
- AWS services are like LEGO blocks customizable
- Event-driven thinking improves app design