AWS Foundation Capstone Project

2-Day Learning Path for Interns & New Employees

Prerequisites

- AWS Account (free tier)
- Basic understanding of web applications
- Text editor installed

DAY 1: Foundation & Core Services

Module 1: Getting Started (30 mins)

Goal: Set up AWS account and understand the console

1. Create AWS Free Tier account at aws.amazon.com
2. Enable MFA (Multi-Factor Authentication) on root account
3. Create an IAM user for yourself with admin access
4. Log in as IAM user (never use root account for daily work)
5. Explore the AWS Console - find Services menu and search bar

Key Takeaway: Always use IAM users, never root account

Module 2: EC2 - Your First Server (90 mins)

Goal: Launch and connect to a virtual server

Steps:

1. Go to EC2 dashboard
2. Click "Launch Instance"
3. Choose Amazon Linux 2 AMI (free tier eligible)
4. Select t3.micro instance type
5. Create a new key pair (download and save it safely)
6. Allow SSH (port 22) in security group
7. Launch instance

8. Connect via EC2 Instance Connect (browser-based)
9. Run these commands:

   ****Update all packages** - sudo yum update –y**

   sudo yum install -y httpd

   sudo systemctl start httpd

   `sudo systemctl enable httpd`

   `sudo systemctl status httpd`

10. Add HTTP (port 80) to security group
11. **Create your first HTML page:**

    ```
    echo "<html>
    <head><title>My First AWS Server</title></head>
    <body style='font-family: Arial; text-align: center; padding-
    top: 50px;'>
      <h1 style='color: #FF9900;'>Hello from AWS EC2</h1>
      <p>This server is running on Amazon EC2</p>
      <p>Created by: <strong>YOUR NAME HERE</strong></p>
      <p>Date: $(date)</p>
    </body>
    </html>" | sudo tee /var/www/html/index.html
    ```

    Replace "YOUR NAME HERE" with your actual name
12. To ensure the changes in the index.html file:

    cat /var/www/html/index.html
13. Visit your instance's public IP in browser

***** Note: After accessing your web server, remove the "s" from the URL (changing `https` to `http`) to verify the website hosted on your EC2 instance. This step helps you understand why the secure protocol (`https`) is being disabled in this case.***

Key Takeaway: EC2 = virtual servers in the cloud

Module 3: S3 - Cloud Storage (60 mins)

Goal: Store and serve files from cloud storage

Steps:

1. Go to S3 console
2. Create a bucket (unique name, default settings)

3. Upload an image file
4. Try to access it via URL (will fail - it's private)
5. Make bucket public for static website hosting:
    a. Go to Permissions tab
    b. Uncheck "Block all public access"
    c. Add bucket policy for public read
6. Enable Static Website Hosting
7. Upload index.html file
8. Access website via S3 endpoint

Sample index.html:

```
<!DOCTYPE html>
<html>
<head><title>My S3 Site</title></head>
<body>
  <h1>Hosted on S3</h1>
  <img src="your-image.jpg" alt="My Image">
</body>
</html>
```

Key Takeaway: S3 = unlimited file storage, can host static websites

Module 4: IAM - Security Basics (45 mins)

Goal: Understand AWS security model

Steps:

1. Go to IAM console
2. Create a new user group "Developers"
3. Attach policy "AmazonS3ReadOnlyAccess"
4. Create a new user and add to "Developers" group
5. Try accessing S3 with this user (read-only)
6. Create a custom policy that allows only listing buckets
7. Review IAM best practices in dashboard

Key Concepts:

- Users = individual people

- Groups = collections of users
- Roles = for AWS services to access other services
- Policies = permissions in JSON format

Key Takeaway: IAM controls who can do what in your AWS account

Module 5: RDS - Managed Database (60 mins)

Goal: Set up a managed MySQL database

Steps:

1. Go to RDS console
2. Click "Create Database"
3. Choose MySQL
4. Select Free Tier template
5. Set master username and password (save these!)
6. Use db.t4g.micro instance
7. Create database
8. Wait 5-10 minutes for creation
9. Click on your database name
10. Under **"Connectivity & security"** tab
11. Click on the VPC security group link (e.g., `rds-sg`)
12. Select the security group
13. Click **"Edit inbound rules"**
14. Click **"Add rule"**
15. Type: **MySQL/Aurora**
16. Port: 3306 (auto-fills)
17. Source: **"Security Group ID"**
18. Click **"Save rules"**
19.
20. Connect from EC2 instance:
```
sudo dnf install -y mariadb105
Mysql --version
mysql -h your-rds-endpoint -u admin -p
```
21. Create a simple table:

-- 1. Create a new database: CREATE DATABASE testdb;

-- 2. Switch to the new database: USE testdb;

-- 3. Create a table named 'users': CREATE TABLE users (id INT, name VARCHAR(50));

-- 4. Insert a sample record: INSERT INTO users VALUES (1, 'Alice');

-- 5. Query all records from the table SELECT * FROM users;

Key Takeaway: RDS = managed databases, no server maintenance needed

DAY 2: Advanced Services & Integration

Module 6: VPC - Networking Basics (45 mins)

Goal: Understand AWS networking

Concepts to explore:

1. Open VPC dashboard
2. Observe your default VPC
3. Examine:
    a. Subnets (public vs private)
    b. Route tables
    c. Internet Gateway
    d. Security Groups vs NACLs

Simple exercise:

- Modify your EC2 security group to only allow SSH from your IP
- Test connection still works
- Change to different IP and verify it fails

Key Takeaway: VPC = your private network in AWS cloud

Module 7: Lambda - Serverless Computing (60 mins)

Goal: Run code without managing servers

Steps:

1. Go to Lambda console
2. Create function "HelloWorld"

3. Choose Python 3.x runtime
4. Replace code with:

```
import json
def lambda_handler(event, context):
    """
    AWS Lambda handler that returns a simple greeting.
    """
    # Get 'name' from event, default to 'World'
    name = event.get('name', 'World')

    # Build response
    response = {
        'statusCode': 200,
        'body': json.dumps(f'Hello {name}!')
    }
    return response
```

5. Test with: `{"name": "AWS Learner"}`
6. Create another function to read from S3:

```
import boto3
import json

def lambda_handler(event, context):
    """
    AWS Lambda handler to list objects in an S3 bucket.
    """
    # Initialize S3 client
    s3 = boto3.client('s3')
    # Define bucket name
    bucket = 'your-bucket-name'
 # List objects in the bucket
    objects = s3.list_objects_v2(Bucket=bucket)

    # Extract file keys (object names)
    files = [obj['Key'] for obj in objects.get('Contents', [])]

    # Build response
    response = {
        'statusCode': 200,
        'body': json.dumps(files)
    }

    return response
```

7. Add S3 read permissions to Lambda role

Key Takeaway: Lambda = run code on-demand, pay only for execution time

Module 8: CloudWatch - Monitoring (45 mins)

Goal: Monitor your AWS resources

Steps:

1. Go to CloudWatch console
2. Check metrics for your EC2 instance (CPU, network)
3. Create an alarm:
    a. Metric: EC2 CPU Utilization
    b. Threshold: > 80%
    c. Action: Send notification to email
4. View Lambda logs under Log Groups
5. Explore CloudWatch dashboards

Key Takeaway: CloudWatch = monitoring and logging for all AWS services

Module 9: API Gateway + Lambda (60 mins)

Goal: Create a REST API

Steps:

1. Go to API Gateway console
2. Create REST API
3. Create resource "/hello"
4. Create GET method
5. Integrate with your HelloWorld Lambda
6. Enable CORS
7. Deploy API to "dev" stage
8. Test the endpoint URL in browser

9. Add POST method to save data

Key Takeaway: API Gateway = create APIs that trigger Lambda functions

Module 10: Final Capstone Project (2-3 hours)

Goal: Build a complete application integrating multiple services

Project: Simple Contact Form Application

Architecture:

- S3: Host static website (HTML form)
- API Gateway: REST endpoint
- Lambda: Process form submission
- DynamoDB: Store contact submissions (free tier)
- CloudWatch: Monitor and log

Steps:

1. Create DynamoDB table "Contacts" with primary key "id"
2. Create Lambda function to save to DynamoDB:

```
import boto3
import json
import uuid
from datetime import datetime

# Initialize DynamoDB resource and table
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('Contacts')


def lambda_handler(event, context):
  """
  AWS Lambda handler to save contact messages into DynamoDB.
  """
  # Parse request body
  body = json.loads(event['body'])

  # Create item to insert
```

```
    item = {
        'id': str(uuid.uuid4()),
        'name': body['name'],
        'email': body['email'],
        'message': body['message'],
        'timestamp': datetime.now().isoformat()
    }

    # Save item to DynamoDB
    table.put_item(Item=item)

    # Return response
    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Origin': '*'
        },
        'body': json.dumps('Message saved!')
    }
```

3. Create API Gateway endpoint connected to Lambda
4. Create HTML form in S3 that calls your API
5. Test the complete flow

Cleanup Checklist (Important!)

To avoid charges, delete these resources:

- Terminate all EC2 instances
- Delete RDS databases
- Empty and delete S3 buckets
- Delete Lambda functions
- Delete API Gateway APIs
- Delete DynamoDB tables
- Delete CloudWatch alarms

Key AWS Concepts Covered

1. **Compute:** EC2, Lambda
2. **Storage:** S3, EBS (with EC2)
3. **Database:** RDS, DynamoDB
4. **Networking:** VPC, Security Groups
5. **Security:** IAM, MFA
6. **Monitoring:** CloudWatch
7. **Integration:** API Gateway

Additional Resources

- AWS Free Tier: aws.amazon.com/free
- AWS Documentation: docs.aws.amazon.com
- AWS Training: aws.amazon.com/training/