AWS Midterm Implementation Guide

1. Hosting Static Websites on AWS S3 and EC2

```
**S3 Website Hosting**
1. Go to S3 Console → Create bucket (unique name, region).
2. Disable Block Public Access (required for website hosting).
3. Upload website files (index.html, error.html).
4. Go to Properties → Static website hosting → Enable → Enter index.html and error.html.
5. Add bucket policy:
{
  "Version":"2012-10-17",
  "Statement":[{
    "Effect":"Allow",
    "Principal":"*",
    "Action":"s3:GetObject",
    "Resource":"arn:aws:s3:::your-bucket-name/*"
  }]
}
6. Copy bucket Endpoint URL → open in browser.

**EC2 Website Hosting**
1. Launch EC2 (Amazon Linux/Ubuntu).
   - Security Group: allow HTTP (80) + SSH (22).
2. Connect via SSH:
   ssh -i "key.pem" ec2-user@<Public-IP>
3. Install Apache:
   sudo yum update -y
   sudo yum install httpd -y
   sudo systemctl start httpd
   sudo systemctl enable httpd
4. Upload website files to /var/www/html/.
   echo "Hello from EC2!" | sudo tee /var/www/html/index.html
5. Access via Public IP in browser.
```

2. EC2 Setup and MySQL Database Management

```
**Install MySQL**
sudo apt update
sudo apt install mysql-server -y
sudo mysql_secure_installation

**Create Database & User**
CREATE DATABASE studentdb;
CREATE USER 'examuser'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON studentdb.* TO 'examuser'@'%';
FLUSH PRIVILEGES;

**Create Table Example**
USE studentdb;
CREATE TABLE students(id INT PRIMARY KEY, name VARCHAR(50));

**Trigger Example**
```

```
CREATE TABLE logs(id INT AUTO_INCREMENT PRIMARY KEY, action VARCHAR(50));
CREATE TRIGGER log_insert
AFTER INSERT ON students
FOR EACH ROW
INSERT INTO logs(action) VALUES('insert');

**Stored Procedure Example**
DELIMITER //
CREATE PROCEDURE GetStudents()
BEGIN
  SELECT * FROM students;
END //
DELIMITER ;
```

3. Web Application Deployment using AWS Elastic Beanstalk

```
**Elastic Beanstalk Deployment (CLI)**
1. Install EB CLI:
   pip install awsebcli

2. Create project folder with your app code (e.g., app.py).
3. Initialize EB:
   eb init -p python-3.8 myapp
4. Create environment:
   eb create myapp-env
5. Deploy app:
   eb deploy
6. Open in browser:
   eb open

**Elastic Beanstalk Console**
1. Go to Elastic Beanstalk Console → Create Application.
2. Upload ZIP of your app.
3. Elastic Beanstalk provisions EC2, Auto Scaling, Load Balancer, S3 automatically.
```

4. Serverless Computing – S3 and Lambda Integration

```
1. Create S3 bucket.
2. Create Lambda function (Python 3.x runtime). Example code:
import json
def lambda_handler(event, context):
    print("Event:", event)
    return {"statusCode": 200, "body": json.dumps("File processed")}

3. Assign IAM role with S3 + CloudWatch permissions.
4. In S3 → Properties → Event Notifications → Add Lambda trigger for s3:ObjectCreated:*.
5. Upload file → Check CloudWatch Logs for Lambda output.
```

5. EC2 Auto Scaling using Launch Templates and Scaling Policies

```
1. Go to EC2 → Launch Templates → Create template.
   - Define AMI, instance type, security group, key pair.
2. Create Auto Scaling Group (ASG).
   - Select launch template, subnets, optional Load Balancer.
```

3. Add Scaling Policy:
   - Target Tracking: keep average CPU at 50%.
   - Step Scaling: add 1 instance if CPU > 70% for 5 min.
   - Scheduled Scaling: scale at specific time.
4. Test by stressing instance (install 'stress' tool) → verify ASG scales automatically.

6. S3 Bucket File Management and Public Access

1. Create S3 bucket.
2. Upload files (console or CLI):
   aws s3 cp myfile.txt s3://mybucket/

3. Manage files:
   aws s3 ls s3://mybucket/
   aws s3 rm s3://mybucket/myfile.txt

4. Configure Public Access:
   - Disable 'Block Public Access'.
   - Add bucket policy (same as Topic 1).
   - Or make file public individually via console.