For a DevOps-based role specializing in AWS Lambda, you'll want to focus on the following key topics:

Key AWS Lambda Topics to Focus On:

1. **Lambda Basics:**
    - What is AWS Lambda?
    - Event-driven architecture
    - AWS Lambda pricing (Pay-as-you-go model)
    - Cold starts and warm invocations

      Cold starts = slower, due to setup and initialization, Warm invocations = faster, reusing existing environment. To mitigate cold start AWS gives "provisioned concurrency," which keeps a set number of execution environments pre-warmed and Minimize heavy initialization logic, such as opening database connections or loading large files, to reduce the impact of cold starts.
    - Limits and quotas (memory, execution timeout, etc.)

      Maximum Timeout: 15 minutes (900 seconds)
      Default: 3 seconds
      Default Limit: 1,000 concurrent executions (can be increased with AWS support)
      Provisioned Concurrency: Ensures warm starts for a set number of concurrent instances.
      Ephemeral Disk Storage: 512 MB (temporary file storage during execution)
      Max File Descriptors/Processes: 1,024 file descriptors, 1,024 processes/threads

2. **Integration with Other AWS Services:**
    - Lambda triggers (S3, SNS, SQS, DynamoDB, CloudWatch, etc.)
    - Lambda with API Gateway
    - Lambda with Step Functions for orchestration
    - Lambda with S3 events, DynamoDB streams, etc.
3. **Deployment & Automation:**
    - AWS SAM (Serverless Application Model) or CloudFormation for deploying Lambda functions
    - Infrastructure as Code (Terraform/CloudFormation) for automating Lambda deployments
    - CI/CD for Lambda with tools like Jenkins, CodePipeline
4. **Lambda Layers:**

- Reusable libraries and dependencies
- Packaging and using layers

5. **Monitoring & Logging:**
   - Lambda integration with CloudWatch for logs and monitoring
   - X-Ray for tracing
   - Alarms and notifications
6. **Security:**
   - IAM roles and policies for Lambda execution
   - VPC integration for Lambda functions
   - Securing Lambda with KMS (Key Management Service)
7. **Advanced Concepts:**
   - Concurrency and throttling
   - Error handling (retry mechanisms, dead-letter queues)
   - Provisioned concurrency for reducing cold starts
   - Lambda destinations (asynchronous invocations)
   - Performance optimization techniques

Basic Template for Writing an AWS Lambda Function (Python):

```
import json

def lambda_handler(event, context):
    # Process the event data
    response = {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
    return response
```

Template Breakdown:

- lambda_handler(event, context): The entry point for your function.
- event: The data passed to your function (triggered by another AWS service).
- context: Provides runtime information like request IDs, timeout, etc.
- Response: The return value from the Lambda function that includes the HTTP status and body.

AWS Lambda Real-Time Project-Based Examples:

1. Cost Optimization:

Goal: Automatically stop idle EC2 instances to save costs.

```
import boto3
def lambda_handler(event, context):
    ec2 = boto3.client('ec2')
    instances = ec2.describe_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])

    for reservation in instances['Reservations']:
        for instance in reservation['Instances']:
            instance_id = instance['InstanceId']
            ec2.stop_instances(InstanceIds=[instance_id])
            print(f'Stopped EC2 instance: {instance_id}')
```

2. Billing Management:

Goal: Send an alert if the AWS monthly cost exceeds a certain threshold.

```
import boto3
import json
from datetime import datetime

def lambda_handler(event, context):
    cloudwatch = boto3.client('cloudwatch')
    billing_alarm = cloudwatch.describe_alarms(AlarmNames=['BillingAlarm'])

    if billing_alarm['MetricAlarms'][0]['StateValue'] == 'ALARM':
        print("Billing threshold exceeded. Notify stakeholders.")
        # You can integrate with SNS for notifications here.
```

3. Resource Cleanup:

Goal: Automatically delete old EBS snapshots that are older than 30 days.

```
import boto3
from datetime import datetime, timedelta

def lambda_handler(event, context):
    ec2 = boto3.client('ec2')
    retention_days = 30
    delete_time = datetime.now() - timedelta(days=retention_days)

    snapshots = ec2.describe_snapshots(OwnerIds=['self'])['Snapshots']
    for snapshot in snapshots:
        snapshot_time = snapshot['StartTime'].replace(tzinfo=None)
```

```
        if snapshot_time < delete_time:
            ec2.delete_snapshot(SnapshotId=snapshot['SnapshotId'])
            print(f'Deleted snapshot: {snapshot["SnapshotId"]}')
```

4. Automated Backup:

Goal: Create daily snapshots for all running EC2 instances.

```
import boto3
def lambda_handler(event, context):
    ec2 = boto3.client('ec2')
    instances = ec2.describe_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])

    for reservation in instances['Reservations']:
        for instance in reservation['Instances']:
            instance_id = instance['InstanceId']
            ec2.create_snapshot(Description=f"Snapshot of {instance_id}", InstanceId=instance_id)
            print(f'Created snapshot for EC2 instance: {instance_id}')
```

5. Log Processing:

Goal: Automatically process and store logs from an S3 bucket to a DynamoDB table.

```
import boto3
def lambda_handler(event, context):
    s3 = boto3.client('s3')
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('LogTable')

    for record in event['Records']:
        bucket = record['s3']['bucket']['name']
        key = record['s3']['object']['key']

        log_data = s3.get_object(Bucket=bucket, Key=key)
        log_content = log_data['Body'].read().decode('utf-8')

        table.put_item(Item={'log_key': key, 'log_data': log_content})
        print(f'Stored log {key} into DynamoDB')
```

6. Monitoring and Alerts:

Goal: Create a Lambda function to monitor CPU usage of EC2 instances and trigger alerts.

```
import boto3
def lambda_handler(event, context):
    cloudwatch = boto3.client('cloudwatch')
    sns = boto3.client('sns')

    alarms = cloudwatch.describe_alarms(
```

```
        AlarmNames=['HighCPUUsage'],
        StateValue='ALARM'
    )

    if alarms['MetricAlarms']:
        sns.publish(TopicArn='arn:aws:sns:region:account-id:topic', Message='High CPU Usage Alert!')
        print("High CPU alert sent!")
```

Categories for Quick Revision:

1. **Cost Optimization:**
    - EC2 Idle Instance Management
    - Unused Resources Cleanup (e.g., EBS snapshots)
2. **Billing Management:**
    - Billing Alerts
    - Cost Threshold Monitoring
3. **Resource Automation:**
    - Automated EC2 Backups
    - Scheduled Scaling & Resource Management
4. **Log Processing:**
    - S3 log data processing
    - Storing logs in DynamoDB
5. **Monitoring & Alerts:**
    - CPU Usage Monitoring
    - Event-driven alerts
6. **Security Automation:**
    - IAM role enforcement
    - VPC network security

By focusing on these topics and examples, you'll be well-prepared for your AWS Lambda-based DevOps interview!