ARCHITECTURE DOCUMENTATION

Proxtera Cloud Engineer Technical Assignment

Abstract

This architecture is designed to support serverless APIs within the AWS cloud environment.

Yi Renjie (Roger)

rogeryi1991@gmail.com

Detailed Architecture Documentation

1. Introduction

1) Purpose:

This architecture is designed to support serverless APIs within the AWS cloud environment.

2) Assumption:

The scenario involves an online education platform requiring the deployment of three distinct APIs in the cloud. The first API facilitates file uploads for administrators. The second API enables users to download files. The third API is used to modify user profiles.

3) Scope:

This document provides a comprehensive overview of the architecture, including architecture detailed descriptions, monitoring and events, security and compliance, further considerations and recommendations.

2. Architecture

1) Overview:

Using a serverless architecture deployed in a single region, primarily leveraging S3 for file storage. CloudFront is used to accelerate access to S3 and API Gateway, and API Gateway is used to manage the APIs that need to be published. The logic of the APIs involves using Lambda to read and write relevant data, returning data or files. Cognito is used for user authorization of API Gateway.

2) Diagram:

The overall architecture diagram is shown in Fig 1. Except for CloudFront, which is a global resource, all other AWS resources are configured in the same region. This ensures that the entire architecture is serverless, eliminating the need for further infrastructure management.

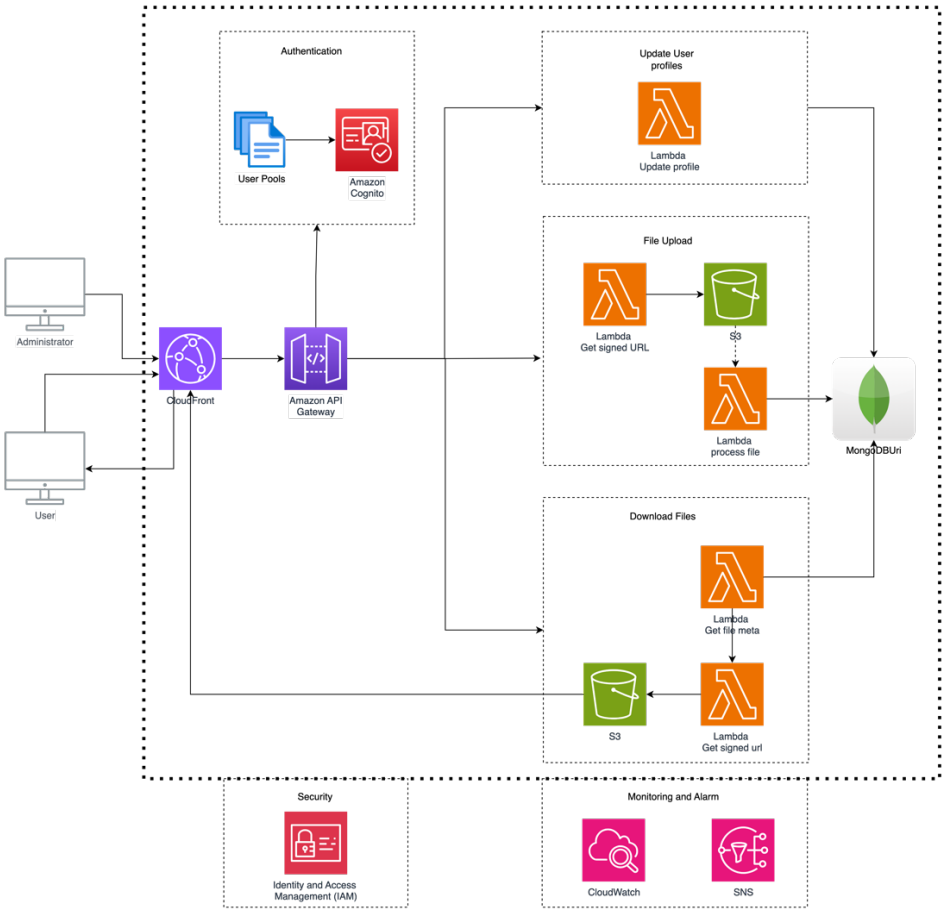

Fig 1. Serverless Architecture

3) **Components:**
   a. **MongoDB Atlas serverless**: Using MongoDB's officially hosted Atlas service can effectively reduce the operational overhead of database management. The serverless mode better fits the overall serverless architecture.
   b. **Lambda**: Use Lambda to access data in MongoDB and perform related operations on files.
   c. **S3**: Used to store files.
   d. **CloudFront**: Accelerates access to S3 files and API Gateway for global users.
   e. **API Gateway**: Manages APIs and triggers Lambda functions to perform related functions.
   f. **Cognito**: Used for user authorization for API access, with a pre-configured user pool.

4) **Infrastructure Architecture Diagram**
   Secure the data flows in VPC
   Use Lambda to securely invoke S3 and MongoDB within the VPC, as shown in Fig 2.

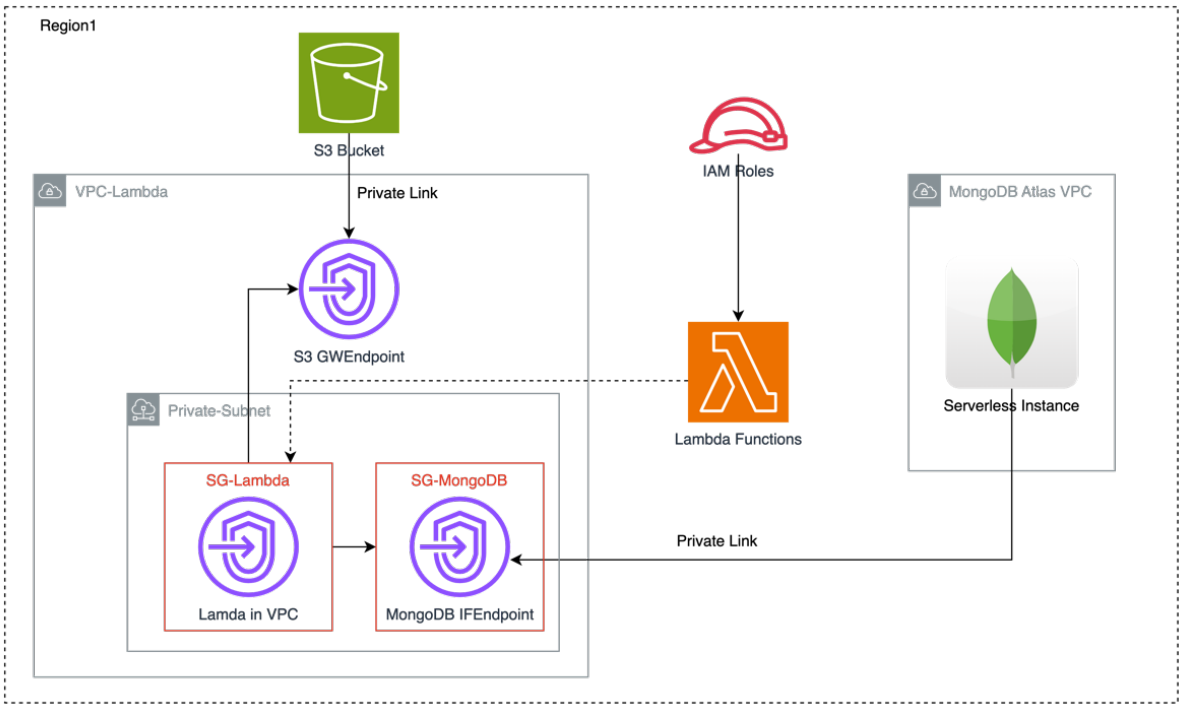

Fig 2. Infrastructure Architecture

a. In this solution, the VPC should not have an Internet Gateway (IGW) or NAT Gateway enabled, and it should use private subnets to deny direct access from the internet.
b. Create a private link to the private subnet for MongoDB Atlas Serverless.
c. Create a private link for the S3 bucket using a Gateway Endpoint and set the route in the route table.
d. Configure the Lambda functions to use the VPC private subnet for invocation.
e. Create two security groups, SG-Lambda and SG-MongoDB. The outbound rule for SG-Lambda allows traffic to the S3 Gateway Endpoint and SG-MongoDB. The inbound rule for SG-MongoDB allows traffic from SG-Lambda.
f. Lambda functions need to use Assumed Roles to invoke S3.

3. Monitoring and Events

1) MongoDB Atlas serverless

MongoDB Atlas provides its own monitoring tools and metrics, which can be integrated into your AWS account by following these steps. Create an IAM Role in your AWS account with the necessary permissions to access CloudWatch. Then, assume this role in MongoDB Atlas and import the relevant monitoring data into CloudWatch in your AWS account.

The provided monitoring data includes:

- **Connections**: Number of active connections. Trigger an alarm if connections exceed 500 times within one minute.
- **Ops/sec**: Number of operations per second (including read, write, delete, etc.). Trigger an alarm if ops/sec exceed 5000 times within one minute.

2) **Lambda**:
   - **Invocations**: Number of times the function is called.
   - **Duration**: Time taken to execute the function.
   - **Errors**: Number of errors encountered during invocation. Trigger an alarm if errors exceed 0 times within one minute.
   - **Throttles**: Number of invocations denied due to concurrency limits. Trigger an alarm if throttles exceed 0 times within one minute.
   - **ConcurrentExecutions**: Number of functions executing concurrently.

3) **S3**:
   - **AllRequests**: Number of all requests.
   - **GetRequests**: Number of GET requests.
   - **PutRequests**: Number of PUT requests.
   - **DeleteRequests**: Number of DELETE requests.
   - **4xxErrors**: Count of 4xx errors. Trigger an alarm if 4xx errors exceed 100 times within one minute.
   - **5xxErrors**: Count of 5xx errors. Trigger an alarm if 5xx errors exceed 100 times within one minute.

4) **CloudFront**:
   - **4xxErrorRate**: Percentage of 4xx errors. Threshold 2% - 5%, trigger alert if last for 5 minutes.
   - **5xxErrorRate**: Percentage of 5xx errors. Threshold 1% - 3%, trigger alert if last for 5 minutes.
   - **TotalErrorRate**: Total error percentage. Threshold 1% - 5%, trigger alert if last for 5 minutes.
   - **OriginLatency**: Latency from CloudFront to the origin server.

5) **API Gateway**:
   - **4XXError**: Count of client errors. Threshold 2% - 5%, trigger alert if last for 5 minutes.
   - **5XXError**: Count of server errors. Threshold 1% - 3%, trigger alert if last for 5 minutes.
   - **Count**: Total number of requests**.** If the expected peak rate is 1000 per minute, you can set an alert for when it exceeds 20% above this rate.
   - **IntegrationLatency**: Time taken for the API Gateway to call the integration service and receive a response.
   - **CacheHitCount**: Number of cache hits.
   - **CacheMissCount**: Number of cache misses.

6) **Coginito**:
   - **SignInSuccesses**: Number of successful sign-ins.
   - **SignInFailures**: Number of failed sign-ins.
   - **SignUpSuccesses**: Number of successful sign-ups.
   - **SignUpFailures**: Number of failed sign-ups.
   - **TokenRefreshSuccesses**: Number of successful token refreshes.

- **TokenRefreshFailures**: Number of failed token refreshes.
- **UserPoolSuccess**: Number of successful user pool operations (e.g., updating user attributes).
- **UserPoolFailure**: Number of failed user pool operations.

7) VPC Logs

Use CloudWatch to monitor VPC flow logs and ensure there is no unauthorized public access to the VPC. You can automate alerts, manually analyze logs, and take appropriate actions to secure your VPC.

- **Destination IP Address (dstaddr)**: Ensure the destination IP address of the traffic is not a public IP.
- **Source IP Address (srcaddr)**: Ensure the source IP address of the traffic is not a public IP.

4. Security and Compliance

1) MongoDB Atlas serverless configurations

a. Add the IP addresses of the Lambda functions in the private subnet to the whitelist in the DB Instance. Ensure that the whitelist does not include any other public IP addresses or 0.0.0.0/0 policies.
b. Use AWS Secrets Manager to manage database keys and rotate them regularly to ensure security and compliance.
c. Serverless automatically encrypts data at rest, but you can use AWS KMS to encrypt data with custom keys and rotate them regularly.

2) Lambda

a. Ensure that any data processed by Lambda functions is encrypted, including data in transit (using HTTPS or TSL) and data at rest.
b. Identify Personally Identifiable Information (PII) data and ensure that PII data is encrypted during function invocation.
c. Integrate CI/CD tools (such as AWS CodePipeline) for automated testing and deployment of code to reduce human error.
d. Publish multiple versions of Lambda functions and use aliases to point to specific versions. This allows for rollback to previous versions and gradual deployment as needed.
e. Use IAM Roles to control that Lambda can only call the resources it is allowed to, such as S3, AWS KMS, AWS Secrets Manager, etc.

3) S3

a. Enable AWS S3 bucket versioning to prevent accidental file overwrites.
b. Set up signed URLs for uploading files.
c. Use server-side encryption provided by AWS Key Management Service (KMS) to enhance the management and control of data encryption.
d. Use HTTPS for data transmission to ensure that data is encrypted during transit.
e. Implement lifecycle management for timely archiving and deletion of relevant objects.

4) **CloudFront**
   a. Use OAI (Origin Access Identity): Ensure that only CloudFront can access your S3 buckets by using Origin Access Identity (OAI), thereby protecting the origin server from direct access.
   b. Enable HTTPS encryption to ensure data is secure during transmission. You can enforce the use of HTTPS and use TLS 1.2 or higher versions.
   c. Use CloudFront's custom SSL/TLS certificates to ensure secure communication.
   d. Set appropriate caching policies based on content sensitivity to avoid caching sensitive data or exposing unnecessary information in the cache.
   e. Configure geographic restrictions to limit access to your content from specific regions, complying with data sovereignty or regional compliance requirements.
   f. Restrict the distribution scope of content as needed, such as allowing only authenticated users to access the content.

5) **API Gateway**
   a. Select the API and click on "Method Request" under "Resources." Click "Method Request," and then find the "Request Validation" section under "Method Request." Add a resource policy to restrict API Gateway access to only allow requests from CloudFront. Ensure that the API Gateway is only accessible by CloudFront in this architecture.
   b. Use Cognito for authentication and authorization.
   c. Set limits on the maximum size of requests and responses to prevent abuse or resource exhaustion.
   d. When handling sensitive data in API requests and responses, ensure that the data is encrypted during both transmission and storage.

6) **Cognito**
   a. **Multi-Factor Authentication (MFA)**: Enable MFA to add an extra layer of security. Supports SMS verification codes and TOTP (Time-Based One-Time Password) applications (such as Google Authenticator) as MFA methods.
   b. **Configure Strong Password Policies**: Implement strong password policies, including requirements for password length, complexity, and expiration, to enhance account security.
   c. **Enable Email and Phone Number Verification**: Ensure that users' identity information is valid and verified by enabling email and phone number verification.
   d. **Configure Account Recovery Options**: Set up account recovery options to ensure users can safely recover their accounts if locked out or if they lose their passwords.

5. Further Considerations and Recommendations

1) **Internet Security and Intrusion Detection**

   In the event of a significant DDoS attack, or related SQL injection and XSS (Cross-Site Scripting) attacks:
   a. Use AWS Shield Advanced and AWS WAF to protect CloudFront.
   b. Traffic should pass through Shield Advanced before reaching WAF.

c. AWS deploys a large number of anti-DDoS devices at the internet entry points of its data centers and collaborates with major telecom operators for traffic cleaning.
d. Cleaned traffic then enters WAF for more refined security protection.

2) Compliance Recommendations

a. Summarize the deployment process, create CloudFormation files according to the process, and generate StackSets. Set up drift detection.
b. Use AWS Config to continuously monitor resource configurations, view compliance reports, and historical records. Reports show the compliance status of resources and help identify and fix non-compliant resources.
c. AWS CloudTrail records all API call activities in AWS accounts and provides auditing capabilities to ensure compliance and security.

3) Disaster Recovery Backup

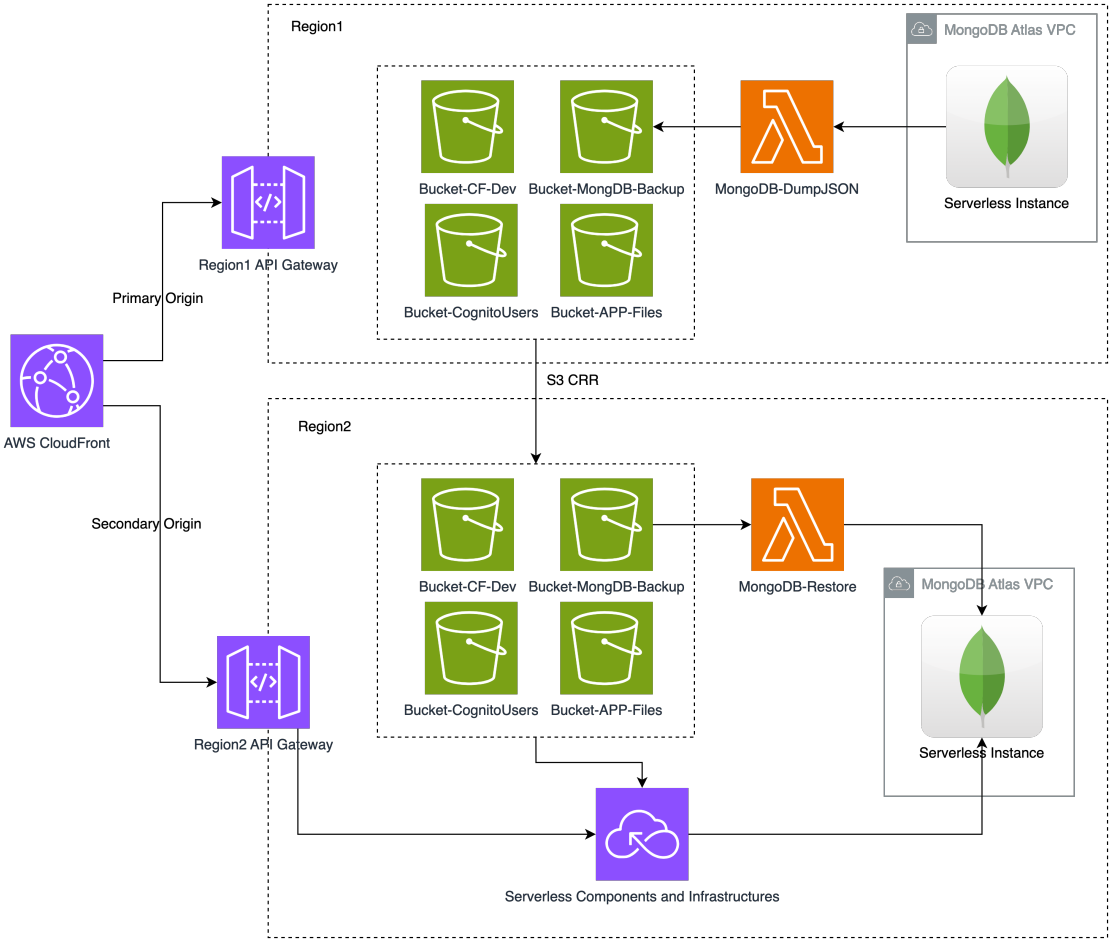

Fig 3. Disaster Recovery Backup

a. MongoDB
Since the MongoDB Atlas serverless automatically creates a snapshot every 6 hours but does not allow downloads, we use EventBridge and Lambda to schedule and trigger a function to connect to MongoDB for backups.
   - Use EventBridge to create a schedule event named MongoDBDumpEvery6Hours.

- Create a Lambda function that utilizes the pymongo library to query the data, convert it to JSON format, and upload it to S3.
- Ensure that during this process, the Lambda function assumes the role of S3 BucketMongoDBBackup and uses AWS Secrets Manager for MongoDB credentials and permissions.

b. S3 Cross-Region Replication
- Use S3 Cross-Region Replication (CRR) to replicate MongoDB backup files and other application files stored in S3 to Region2.
- Store user information saved in Cognito in S3, ensuring the data is encrypted. Also, enable S3 CRR for this data.
- Store all deployment-related CloudFormation stack sets in S3, including different versions of Lambda functions, and enable S3 CRR for these files as well.

c. Failover and Recovery
- Configure primary and secondary origins in the CDN. For example, set the API Gateway in Region1 as the primary origin and another in Region2 as the secondary origin. Define failover conditions based on monitoring and alerts.
- In the event of a failure, use CloudFormation to quickly set up the necessary resources in Region2 and ensure business continuity. Make sure to restore the latest MongoDB backup data.
- Note that this solution is feasible and efficient if higher RTO (Recovery Time Objective) and RPO (Recovery Point Objective) are not required.

4) VPC Interface Endpoint Single Point of Failure

This infrastructure architecture provides a higher availability, as shown in Fig 4.

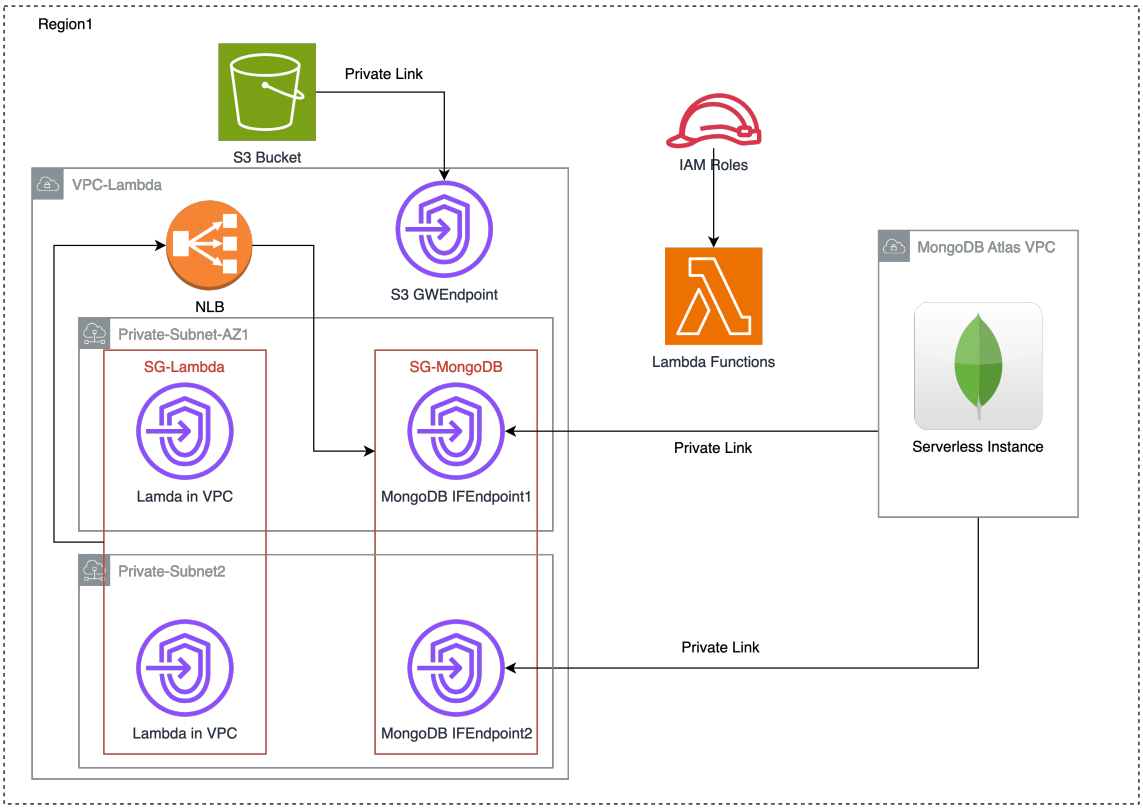

Fig 4. Higher Availability in two AZs

a. Create two interface endpoints for MongoDB Atlas Serverless.
b. Deploy these endpoints in private subnets located in different Availability Zones (AZs).
c. Create an NLB (Network Load Balancer) instance in this VPC and designate the target group as these two endpoints.
d. Configure the NLB to use the least connections load balancing algorithm and set up health checks.
e. Configure Lambda functions to run in these two private subnets and call the NLB Listener.
f. Add new endpoints to SG-Lambda and SG-MongoDB.