Replaying Parallel Requests to Break a Monolith

Use this architecture to break your monolith with confidence by setting up a parallel run strategy combined with the strangler fig pattern. First proxy the methods to be replaced with a microservice, storing a copy of user requests and monolith responses in a time-series database. Then replay the requests on your new microservice to compare the responses between the legacy monolith and the replacement microservice.

1. **AWS Migration Hub Refactor Spaces**
   - Users
   - Amazon API Gateway
   - Strangler fig
   - Recorder system
   - AWS Lambda

2. **AWS Step Functions**
   - Manual replay flow
   - Reset
   - Load
   - Microservice temporary DB
   - Amazon RDS database

3. **Amazon S3 monolith responses**
   - Amazon Kinesis Data Firehose S3 connector
   - Amazon Kinesis Data Streams data stream

4. **AWS Lambda sorter function**
   - Insert
   - Amazon Timestream ordered requests

5. **AWS Lambda sequencer function**
   - Push
   - Amazon SQS FIFO queue

6. **AWS Lambda request repeater**
   - Invoke
   - Amazon S3 microservice responses
   - Repeat flow
   - Amazon SNS alarm
   - AWS Lambda response comparer

7. **From the last S3 bucket, prompt a function to compare the responses to the same request sent to the monolith and to the microservice. Raise an alarm for any differences using Amazon Simple Notification Service (Amazon SNS), and store the final results in the requests database.**

8. **Use an AWS Step Functions workflow to replay the requests, first resetting the microservice’s temporary databases from a monolith’s database backup, then replaying the requests into the microservice from the date/time of the backup.**

9. **For each request in the queue, invoke the same request in the microservice’s API, recording the responses in an S3 bucket.**

10. **Assuming a legacy cloud monolith standing on Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Relational Database Service (Amazon RDS), apply the strangler fig pattern with AWS Migration Hub Refactor to place an API in front of your monolith and re-route the endpoints-to-modernize into a recorder system that records all requests and responses.**

11. **Use a proxy recorder AWS Lambda function to proxy requests back to the legacy monolith, and push a copy of all request-and-response payloads into Amazon Kinesis Data Streams.**

12. **Use Amazon Kinesis Data Firehose to deliver the monolith-payload stream into an Amazon Simple Storage Service (Amazon S3) bucket.**

13. **Use a sorter function to store the payloads in Amazon Timestream in a time-based order.**

14. **Using AWS Backup, periodically back up the monolith’s database. This baselines the replay flow’s start time to replay payloads.**

15. **Reset the microservice’s temporary databases from the monolith’s database backup, then replaying the requests into the microservice from the date/time of the backup.**

16. **Use an AWS Step Functions workflow to replay the requests, first resetting the microservice’s temporary databases from a monolith’s database backup, then replaying the requests into the microservice from the date/time of the backup.**

17. **Fetch all requests in the replay-time window, sorted by request date/time, then push them to a first-in-first-out (FIFO) queue using Amazon Simple Queue Service (Amazon SQS).**

18. **For each request in the queue, invoke the same request in the microservice’s API, recording the responses in an S3 bucket.**

19. **From the last S3 bucket, prompt a function to compare the responses to the same request sent to the monolith and to the microservice. Raise an alarm for any differences using Amazon Simple Notification Service (Amazon SNS), and store the final results in the requests database.**

AWS Reference Architecture