Serverless Architectural Patterns and Best Practices

Sascha Möllering

29.03.2017
Agenda

Serverless characteristics and practices
3-tier web application
Batch processing
Stream processing
Operations automation
Wrap-up/Q&A
Spectrum of AWS offerings

“On EC2”
- Amazon EC2

Managed
- Amazon EMR
- Amazon Elasticsearch Service
- Amazon ElastiCache
- Amazon Redshift
- Amazon RDS

Serverless
- AWS Lambda
- Amazon Cognito
- Amazon Kinesis
- Amazon S3
- Amazon DynamoDB
- Amazon SQS
- Amazon API Gateway
- Amazon CloudWatch
- AWS IoT
Serverless patterns built with functions

Functions are the unit of deployment and scale
Scales per request—users cannot over or under-provision
Never pay for idle
Skip the boring parts; skip the hard parts
Lambda considerations and best practices

AWS Lambda is stateless—architect accordingly

• Assume no affinity with underlying compute infrastructure
• Local filesystem access and child process may not extend beyond the lifetime of the Lambda request
Lambda considerations and best practices

Can your Lambda functions survive the cold?

- Instantiate AWS clients and database clients outside the scope of the handler to take advantage of connection re-use.
- Schedule with CloudWatch Events for warmth.
- ENIs for VPC support are attached during cold start.

```python
import sys
import logging
import rds_config
import pymysql

rds_host = "rds-instance"
db_name = rds_config.db_name
try:
    conn = pymysql.connect()
except:
    logger.error("ERROR: Unable to connect to database.")
def handler(event, context):
    with conn.cursor() as cur:
        # Executed with each invocation
```

Executes during cold start
Executes with each invocation
Lambda considerations and best practices

How about a file system?

- Don’t forget about /tmp (512 MB scratch space)

```javascript
exports.ffmpeg = function(event, context) {
  new ffmpeg('./thumb.MP4', function (err, video) {
    if (!err) {
      video.fnExtractFrameToJPG('/tmp')
      function (error, files) {
        if (!error) {
          console.log(files);
          context.done();
        }
      }(...}
    }
  }
};
```
Lambda considerations and best practices

Custom CloudWatch metrics

- 40 KB per POST
- Default Acct Limit of 150 TPS
- Consider aggregating with Kinesis

```python
def put_cstate(iid, state):
    response = cwclient.put_metric_data(
        Namespace='AWSx/DirectConnect',
        MetricData=[
            {
                'MetricName': 'ConnectionState',
                'Dimensions': [
                    {
                        'Name': 'ConnectionId',
                        'Value': iid
                    },
                ],
                'Value': state,
                'Unit': 'None'
            }
        ]
    )
```
Pattern 1: 3-Tier Web Application
Web application

Browser

Amazon CloudFront

Amazon API Gateway

Dynamic content in AWS Lambda

Data stored in Amazon DynamoDB
Serverless web app security

- **Amazon CloudFront**
  - OAI
  - Geo-Restriction
  - Signed Cookies
  - Signed URLs
  - DDOS

- **Amazon S3**
  - Bucket Policies
  - ACLs

- **Amazon API Gateway**
  - Throttling
  - Caching
  - Usage Plans

- **AWS Lambda**

- **Amazon DynamoDB**

**Browser**

**Static Content**

**AuthZ**
Serverless web app security

- Static Content
  - Amazon CloudFront
    - OAI
    - Geo-Restriction
    - Signed Cookies
    - Signed URLs
    - DDOS
  - Amazon S3
    - Bucket Policies
    - ACLs

- Browser

- AWS WAF
  - HTTPS
  - Disable Host Header Forwarding

- Amazon API Gateway
  - Throttling
  - Caching
  - Usage Plans

- AWS Lambda

- Amazon DynamoDB

- IAM

- AuthZ
Serverless web app monitoring

- **Amazon CloudFront**
  - Access Logs in S3 Bucket
  - CloudWatch Metrics: [https://aws.amazon.com/cloudfront/reporting/](https://aws.amazon.com/cloudfront/reporting/)

- **AWS S3**
  - Access Logs in S3 Bucket

- **AWS Lambda**
  - Invocations
  - Invocation Errors
  - Duration
  - Throttled Invocations
  - Latency
  - Throughput
  - Throttled Reqs
  - Returned Bytes

- **AWS WAF**
  - WebACL Testing
  - Total Requests
  - Allowed/Blocked Requests by ACL

- **AWS DynamoDB**
  - Latency
  - Count
  - Cache Hit/Miss
  - 4XX/5XX Errors

- **AWS CloudTrail**

- **Custom CloudWatch Metrics & Alarms**

- **Browser**

- **Static Content**
Serverless web app lifecycle management

AWS SAM (Serverless Application Model) - blog

- Code/Packages/Swagger
- Serverless Template
- Package & Deploy
- Amazon S3
- Serverless Template w/ CodeUri
- AWS CloudFormation
- AWS Lambda
- Amazon DynamoDB
- Amazon API Gateway

package

deploy

CI/CD Tools
Amazon API Gateway best practices

Use mock integrations
Signed URL from API Gateway for large or binary file uploads to S3
Asynchronous calls for Lambda > 30s
Greedy variable, ANY method, proxy integration

Simple yet very powerful:

• Automatically scale to meet demand
• Only pay for the requests you receive
Pattern 2: Batch Processing
Characteristics

Large data sets
Periodic or scheduled tasks
Extract Transform Load (ETL) jobs
Usually non-interactive and long running
Many problems fit MapReduce programming model
Serverless batch processing

- **Amazon S3 Object**
- **AWS Lambda: Splitter**
- **AWS Lambda: Mappers**
- **Amazon DynamoDB: Mapper Results**
- **AWS Lambda: Reducer**
- **Amazon S3 Results**
Considerations and best practices

Cascade mapper functions
Lambda languages vs. SQL
Speed is directly proportional to the concurrent Lambda function limit
Use DynamoDB/ElastiCache/S3 for intermediate state of mapper functions
Lambda MapReduce Reference Architecture
Cost of serverless batch processing

200 GB normalized Google Ngram data-set

Serverless:
  • 1000 concurrent Lambda invocations
  • Processing time: 9 minutes
  • Cost: $7.06
Pattern 3: Stream Processing
Stream processing characteristics

- High ingest rate
- Near real-time processing (low latency from ingest to process)
- Spiky traffic (lots of devices with intermittent network connections)
- Message durability
- Message ordering
Serverless stream processing architecture

KPL: Producer

Amazon Kinesis: Stream

Lambda: Stream Processor

S3: Intermediate Aggregated Data

CloudWatch Events: Trigger every 5 minutes

Lambda: Scheduled Dispatcher

Lambda: Periodic Dump to S3

S3: Final Aggregated Output
Fan-out pattern

- Number of Amazon Kinesis Streams shards corresponds to concurrent Lambda invocations
- Trade higher throughput & lower latency vs. strict message ordering
Best practices

• Tune batch size when Lambda is triggered by Amazon Kinesis Streams – reduce number of Lambda invocations

• Tune memory setting for your Lambda function – shorten execution time

• Use KPL to batch messages and saturate Amazon Kinesis Stream capacity
Monitoring

Amazon Kinesis Stream metric GetRecords.IteratorAgeMilliseconds maximum
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM "device_id",
FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE) as "round_ts",
SUM("measurement") as "sample_sum",
COUNT(*) AS "sample_count"
FROM "SOURCE_SQL_STREAM_001"
GROUP BY "device_id", FLOOR("SOURCE_SQL_STREAM_001".ROWTIME TO MINUTE);
Cost comparison - assumptions

• Variable message rate over 6 hours
• Costs extrapolated over 30 days
## Cost comparison

### Serverless
- Amazon Kinesis Stream with 5 shards

<table>
<thead>
<tr>
<th>Service</th>
<th>Monthly Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Kinesis Streams</td>
<td>$58.04</td>
</tr>
<tr>
<td>AWS Lambda</td>
<td>$259.85</td>
</tr>
<tr>
<td>Amazon S3 (Intermediate Files)</td>
<td>$84.40</td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
<td>$4.72</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$407.01</strong></td>
</tr>
</tbody>
</table>

### Server-based on EC2
- Kafka cluster (3 x m3.large)
- Zookeeper cluster (3 x m3.large)
- Consumer (1 x c4.xlarge)

<table>
<thead>
<tr>
<th>Service</th>
<th>Monthly Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC2 Kafka Cluster</td>
<td>$292.08</td>
</tr>
<tr>
<td>EC2 Zookeeper Cluster</td>
<td>$292.08</td>
</tr>
<tr>
<td>EC2 Consumer</td>
<td>$152.99</td>
</tr>
<tr>
<td><strong>Total On-Demand</strong></td>
<td><strong>$737.15</strong></td>
</tr>
<tr>
<td>1-year All Upfront RI</td>
<td><strong>$452.42</strong></td>
</tr>
</tbody>
</table>
Pattern 4: Automation
Automation characteristics

• Respond to alarms or events
• Periodic jobs
• Auditing and Notification
• Extend AWS functionality

…All while being Highly Available and Scalable
Automation: dynamic DNS for EC2 instances

Amazon EC2 Instance State Changes → Amazon CloudWatch Events: Rule Triggered → AWS Lambda: Update Route53 → Amazon Route53: Private Hosted Zone

Tag:
CNAME = ‘xyz.example.com’

xyz.example.com A 10.2.0.134
Automation: image thumbnail creation from S3

Users upload photos

S3: Source Bucket

AWS Lambda: Resize Images

Triggered on PUTs

S3: Destination Bucket
CapitalOne Cloud Custodian

Amazon CloudWatch Events: Rules Triggered

AWS Lambda: Policy & Compliance Rules

Amazon SNS: Alert Notifications

AWS CloudTrail: Events

Amazon CloudWatch Logs: Logs

Read more here: http://www.capitalone.io/cloud-custodian/docs/index.html
Best practices

• Document how to disable event triggers for your automation when troubleshooting

• Gracefully handle API throttling by retrying with an exponential back-off algorithm (AWS SDKs do this for you)

• Publish custom metrics from your Lambda function that are meaningful for operations (e.g. number of EBS volumes snapshotted)
Thank you!