Serverless Architectural Patterns

Pawan Puthran
Technical Account Manager

pputhran@amazon.com
@pawanputhran
pputhran
Agenda

• Serverless Foundation
• Web application
• Data Lake
• Stream processing
• Operations automation
A serverless world...

“Build and run applications without thinking about servers

... pay per request not for idle”

Scales with usage
High availability built-in
Never pay for idle
No servers to provision or manage
## Building blocks for serverless applications

<table>
<thead>
<tr>
<th>Compute</th>
<th>Storage</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>Amazon S3</td>
<td>Amazon DynamoDB</td>
</tr>
<tr>
<td>AWS Lambda@Edge</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API Proxy and GraphQL</th>
<th>Messaging and Queues</th>
<th>Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon API Gateway</td>
<td>Amazon SQS</td>
<td>Amazon Kinesis</td>
</tr>
<tr>
<td>AWS AppSync</td>
<td>Amazon SNS</td>
<td>Amazon Athena</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orchestration and State Management</th>
<th>Monitoring and Debugging</th>
<th>User Management and IdP</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Step Functions</td>
<td>AWS X-Ray</td>
<td>Amazon Cognito</td>
</tr>
</tbody>
</table>
AWS X-Ray Integration with Serverless

- Lambda instruments incoming requests for all supported languages
- Lambda runs the X-Ray daemon on all languages with an SDK

```javascript
const AWSXRay = require('aws-xray-sdk-core');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
const AWS = AWSXRay.captureAWS(require('aws-sdk'));
S3Client = AWS.S3();
```
X-Ray Trace Example

```
<table>
<thead>
<tr>
<th>Method</th>
<th>Response</th>
<th>Duration</th>
<th>Age</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>202</td>
<td>2.0 sec</td>
<td>1.3 min (2017-04-14 00:42:54 UTC)</td>
<td>1-58f01b0e-53eef2bd463eeecfd7f311ce4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Res.</th>
<th>Duration</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>s3example</td>
<td>202</td>
<td>87.0 ms</td>
<td>✓</td>
</tr>
<tr>
<td>Dwell Time</td>
<td>-</td>
<td>186 ms</td>
<td>✓</td>
</tr>
<tr>
<td>Attempt #1</td>
<td>200</td>
<td>1.8 sec</td>
<td>✓</td>
</tr>
</tbody>
</table>

s3example AWS::Lambda

- s3example: 863 ms ✓
- Initialization: 334 ms ✓
- S3: 762 ms ❌
```

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
AWS Serverless Application Model (SAM)

- CloudFormation extension optimized for serverless
- New serverless resource types: functions, APIs, and tables
- Supports anything CloudFormation supports
- Open specification (Apache 2.0)
  - SAM Translator recently open sourced.

https://github.com/awslabs/serverless-application-model
SAM CLI

- Develop and test Lambda locally
- Invoke functions with mock serverless events
- Local template validation
- Local API Gateway with hot reloading

https://github.com/awslabs/aws-sam-cli
Delivery via CodePipeline

**Pipeline flow:**

1. Commit your code to a source code repository
2. Package/test in CodeBuild
3. Use CloudFormation actions in CodePipeline to create or update stacks via SAM templates
   - **Optional**: Make use of ChangeSets
4. Make use of specific stage/environment parameter files to pass in Lambda variables
5. Test our application between stages/environments
   - **Optional**: Make use of manual approvals
AWS CodeDeploy and Lambda Canary Deployments

- Direct a portion of traffic to a new version
- Monitor stability with CloudWatch
- Initiate rollback if needed
- Incorporate into your SAM templates
Lambda Best Practices

- **Minimize** package size to necessities
- Separate the **Lambda handler** from core logic
- Use **Environment Variables** to modify operational behavior
- Self-contain **dependencies** in your function package
- Leverage “**Max Memory Used**” to right-size your functions
- Delete large **unused** functions (75GB limit)
Pattern 1:
Web App/Microservice/API
Bustle Achieves 84% Cost Savings with AWS Lambda

With AWS Lambda, we eliminate the need to worry about operations

Tyler Love
CTO, Bustle

• Bustle had trouble scaling and maintaining high availability for its website without heavy management
• Moved to serverless architecture using AWS Lambda and Amazon API Gateway
• Experienced approximately 84% in cost savings
• Engineers are now focused on innovation

Bustle is a news, entertainment, lifestyle, and fashion website targeted towards women.
Serverless web app security

- Amazon CloudFront
  - Origin Access Identity (OAI)
  - Geo-Restriction
  - Signed Cookies
  - Signed URLs
  - DDOS Protection

- Amazon S3
  - Bucket Policies
  - ACLs

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

- AWS Lambda

- Amazon DynamoDB

- Amazon Cognito

Browser

Static Content

AuthZ
Custom Authorizers

Two types:
- TOKEN - authorization token passed in a header
- REQUEST – all headers, query strings, paths, stage variables or context variables.
Multi-Region with API Gateway

Client

Amazon Route 53

api.mycorp.com

Custom Domain Name

Regional API Endpoint

API Gateway

Lambda
eu-west-1

eu-central-2

CNAME
Useful Frameworks for Serverless Web Apps

- **AWS Chalice**
  Python Serverless Framework
  https://github.com/aws/chalice
  Familiar decorator-based API similar to Flask/Bottle
  Similar to third-party frameworks, Zappa or Claudia.js

- **AWS Serverless Express**
  Run Node.js Express apps
  https://github.com/awslabs/aws-serverless-express

- **Java - HttpServlet, Spring, Spark and Jersey**
  https://github.com/awslabs/aws-serverless-java-container
Pattern 2: Data Lake
Serverless Data Lake Characteristics

- Collect/Store/Process/Consume and Analyze all organizational data
- Structured/Semi-Structured/Unstructured data
- AI/ML and BI/Analytical use cases
- Fast automated ingestion
- Schema on Read
- Complementary to EDW
- Decoupled Compute and Storage
The Foundation...Amazon S3

- No need to run compute clusters for storage
- Virtually unlimited number of objects and volume
- Very high bandwidth – no aggregate throughput limit
- Multiple storage classes
- Versioning
- Encryption

- AWS CloudTrail Data Events
- S3 Analytics and Inventory
- AWS Config automated checks
- S3 Object Tagging
- S3 Select (NEW!)
- Glacier Select (NEW!)
Search and Data Catalog

- DynamoDB as Metadata repository
- Amazon Elasticsearch

Instantly query your data lake on Amazon S3

- S3 Bucket(s)
- AWS Glue Crawlers
- AWS Glue Data Catalog
- Amazon Athena
- Amazon QuickSight
- Amazon Redshift Spectrum

Catalog & Search

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Analytics and Processing

- Amazon QuickSight
- Amazon Athena
- AWS Lambda
- Predictive Analytics
- Amazon EMR
- AWS Glue (ETL)
Athena – Serverless Interactive Query Service

```
SELECT gram, year, sum(count) FROM ngram
WHERE gram = 'just say no'
GROUP BY gram, year ORDER BY year ASC;
```

44.66 seconds...Data scanned: 169.53GB
Cost: $5/TB or $0.005/GB = $0.85
Athena – Best Practices

• Partition data
  s3://bucket/flight/parquet/year=1991/month=1/day=2/

• Columnar formats – Apache Parquet, AVRO, ORC

• Compress files with splittable compression (bzip2)

• Optimize file sizes

Serverless batch processing

AWS Lambda: Splitter
Amazon S3 Object

AWS Lambda: Mappers

AWS Lambda: Reducer
Amazon S3 Results

Amazon DynamoDB: Mapper Results

Analytics & Processing

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
Financial Modeling is a Monte-Carlo simulation process to project future cash flows, which is used for managing the mortgage risk on daily basis:

- Underwriting and valuation
- Risk management
- Financial reporting
- Loss mitigation and loan removal

- ~10 Quadrillion ($10 \times 10^{15}$) of cash flow projections each month in hundreds of economic scenarios.
- One simulation run of ~ 20 million mortgages takes 1.4 hours, >4 times faster than the existing process.

Federal National Mortgage Association
Pywren

• Pywren Python library provides 10 TFLOPS of peak compute power with new default – 1,000 concurrent functions
• Achieve over 60 GB/sec of read and 50 GB/sec of write performance using Amazon S3

https://pywren.io
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
Streaming data ingestion

AWS Lambda: Transformations & enrichment

Amazon DynamoDB: Lookup tables

Amazon Kinesis Firehose: Delivery stream

Amazon CloudWatch: Delivery metrics

AWS Lambda: Transformations & enrichment

Amazon CloudWatch: Delivery metrics

Amazon S3: Buffered files

Amazon Redshift: Table loads

Amazon Elasticsearch Service: Domain loads

Amazon S3: Source record backup

Record Producers

Kinesis Agent

Raw records

Transformed records

Lookup

Lookup
Best practices

• Tune Firehose buffer size and buffer interval
  • Larger objects = fewer Lambda invocations, fewer S3 PUTs

• Enable compression to reduce storage costs

• Enable Source Record Backup for transformations
  • Recover from transformation errors

• Follow Amazon Redshift Best Practices for Loading Data
  • How to handle time series, sorted data
Sensor data collection

- **IoT Sensors**
  - AWS IoT: Data collection
    - IoT rules
    - IoT actions
  - MQTT

- **Amazon S3:**
  - Raw records

- **Amazon Kinesis Streams:**
  - Real-time stream

- **Amazon Kinesis Firehose:**
  - Delivery stream
  - Batched records

- **Real-time analytics applications**
Real-time analytics

Amazon Kinesis Streams:
- Ingest stream

Amazon Kinesis Analytics:
- Time window aggregation

Amazon Kinesis Streams:
- Aggregates stream

Amazon Kinesis Firehose:
- Error stream

Amazon S3:
- Error records

AWS Lambda:
- Alert function

AWS SNS:
- Notifications

Amazon DynamoDB:
- Device thresholds
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE) as "window_ts",
SUM("measurement") as "sample_sum",
COUNT(*) AS "sample_count"
FROM "SOURCE_SQL_STREAM_001"
GROUP BY "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE);
Real-time analytics

Record Producers

Amazon Kinesis Streams: Ingest stream

Amazon Kinesis Analytics: Time window aggregation

Amazon Kinesis Streams: Aggregates stream

AWS Lambda: Alert function

Amazon DynamoDB: Device thresholds

Amazon Kinesis Firehose: Error stream

AWS SNS: Notifications

Amazon S3: Error records
Amazon Kinesis Streams and AWS Lambda

- Number of Amazon Kinesis Streams **shards** corresponds to **concurrent invocations** of Lambda function
- **Batch size** sets maximum number of records per Lambda function invocation
Fan-out pattern trades strict message ordering vs higher throughput & lower latency

Increase throughput, reduce processing latency
Thomson Reuters – Product Insight

Solution for usage analysis tracking:
Capture, analyze, and visualize analytics data generated by offerings, providing insights to help product teams continuously improve the user experience

Throughput: Tested 4,000 requests / second
Growing to 10,000 requests / second or 25 Billion requests / month

Latency: new events to user dashboards in less than 10 seconds

Durable: no data loss since inception
Best practices

• Tune **batch size** when Lambda is triggered by Amazon Kinesis Streams
  • Higher batch size = fewer Lambda invocations

• Tune **memory** setting for your Lambda function
  • Higher memory = shorter execution time

• Use Kinesis Producer Library (**KPL**) to batch messages and saturate Amazon Kinesis Stream capacity
Pattern 4: Operations Automation
Automation characteristics

• Periodic jobs
• Event triggered workflows
• Enforce security policies
• Audit and notification
• Respond to alarms
• Extend AWS functionality

… All while being Highly Available, Scalable and Auditable
AWS Ops Automator

Amazon DynamoDB: Task configuration & tracking

Amazon CloudWatch: Time-based events

AWS Lambda: Event handler

AWS Lambda: Task executors

Amazon CloudWatch: Logs

AWS SNS: Error and warning notifications

Amazon EC2 Instances

Amazon Redshift Clusters

Resources in multiple AWS Regions and Accounts

Image recognition and processing

1. Start state machine execution
2. Extract image meta-data
3. Generate image thumbnail
4. Store meta-data and tags

Amazon Cognito: User authentication
Amazon S3: Image uploads
AWS Step Functions: Workflow orchestration
Amazon DynamoDB: Image meta-data & tags
Amazon Rekognition: Object detection

https://github.com/awslabs/lambda-refarch-imagerecognition
Step Functions state machine
Enforce security policies

- **RDP from 0.0.0.0/0**
- **New Security Group ingress rule**
- **Amazon CloudWatch Events: Rule**
- **CloudWatch Event Bus in another AWS Account**
- **AWS Lambda: Remediate and alert**
- **AWS SNS: Email alert**
- **Ingress rule deleted**
Autodesk - Tailor

Serverless AWS Account Provisioning and Management Service:

- Automates AWS Account creation,
- Configures IAM, CloudTrail, AWS Config, Direct Connect, and VPC
- Enforces corporate standards
- Audit for compliance

Provisions new Accounts in 10 minutes vs 10 hours in earlier manual process

Open source and extensible: https://github.com/alanwill/aws-tailor
Best practices

• 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)

• Enable **X-Ray** tracing for your Lambda functions

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

Use DevOps tools to **automate** your serverless deployments

Apply serverless patterns for common use-cases:
- Web application
- Data Lake Foundation
- Stream processing
- Operations automation

What will **you** build with Serverless?
Further Reading

- Optimizing Enterprise Economics with Serverless Architectures
  https://d0.awsstatic.com/whitepapers/optimizing-enterprise-economics-serverless-architectures.pdf

- Serverless Architectures with AWS Lambda

- Serverless Applications Lens - AWS Well-Architected Framework

- Streaming Data Solutions on AWS with Amazon Kinesis

- AWS Serverless Multi-Tier Architectures
Thank You!

Pawan Puthran
Technical Account Manager

pputhran@amazon.com
@pawanputhran
pputhran