Getting Started with Serverless Architectures

Miguel Rojo Rossi, Solutions Architect

@MiguelRojoRossi #AWSSummit
Agenda

- Background
- AWS Lambda
- Amazon API Gateway
- Serverless architecture patterns
- Serverless best practices
- Serverless reference architectures
How serverless architecture patterns with AWS Lambda are the next evolution of application design
The Monolithic Architecture
The Monolithic Architecture

- Lots of collateral damage
  - All-for-one and one-to-fail
  - Slipped timelines
  - Operational issues

- Deploy less frequently
  - Less disruption
  - More time to plan

Reduce risk
The Monolithic Organization

Requirements
Prioritization
Scheduling
Deployment
Hardware

“...organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations”

- Conway’s Law
Break it apart.
The Monolithic Architecture
The Service-Oriented Architecture
The Microservices Architecture
Tools to Help This Pattern Are VAST

Web servers
Code libraries
Web service/application frameworks
Configuration management tools
API management platforms
Deployment patterns
CI/CD patterns
Containers
etc. etc. etc.
AWS Has Helped Too!

Amazon EC2
Auto Scaling
Elastic Load Balancing
Auto recovery in EC2
AWS Trusted Advisor
AWS Elastic Beanstalk
AWS OpsWorks
Amazon EC2 Container Service
etc. etc. etc.
But…. 

…many of these tools and innovations are still coupled to a shared dependency.
Servers

(AAHHHHHHHHHHH!!)

How will the application handle server hardware failure?

How can I tell if a server has been compromised?

How can I increase utilization of my servers?

How should I implement dynamic configuration changes on my servers?

Which OS should my servers run?

How much remaining capacity do my servers have?

How will I keep my server OS patched?

How will new code be deployed to my servers?

What size server is right for my performance?

How many servers should I budget for?

How many users create too much load for my servers?

Should I tune OS settings to optimize my application?

When should I decide to scale out my servers?

Which packages should be baked into my server images?

What size servers are right for my budget?

How should I control access from my servers?

When should I decide to scale up my servers?

How can I control access from my servers?

How will I control access from my servers?
Architect to be Serverless

Fully managed
• No provisioning
• Zero administration
• High availability

Developer productivity
• Focus on the code that matters
• Innovate rapidly
• Reduce time to market

Continuous scaling
• Automatically
• Scale up and scale down
Enter AWS Lambda

Serverless, event-driven compute service

Lambda = microservice without servers
Components of Lambda

- A Lambda function (that you write)
- An event source
- AWS Lambda
- The function networking environment
The Lambda Function

- Your code
  (NodeJS, Python, Java)
- The IAM role that code assumes during execution
- The amount of memory allocated to your code (affects CPU and network as well)

A valid, complete Lambda function

```javascript
var AWS = require('aws-sdk');
var s3 = new AWS.S3();

exports.handler = function(event, context) {
    var params = {
        Bucket: '[input bucket name here]',
        Key: '[insert keyname here]',
        Body: '[object body]'  
    };

    s3.putObject(params, function(err, data) {
        if (err) {
            console.log(err, err.stack); // an error occurred
        } else {
            console.log(data);
        }
    context.done();
    });
};
```
An Event Source

Many AWS services can be an event source today:

- Amazon S3
- Amazon Kinesis
- Amazon SNS
- Amazon DynamoDB
- Amazon CloudWatch
- Amazon Config Rules
- …and Amazon API Gateway (more later)
AWS Lambda

- Runs your function code without you managing or scaling servers.
- Provides an API to trigger the execution of your function.
- Ensures function is executed when triggered, in parallel, regardless of scale.
- Provides additional capabilities for your function (logging, monitoring).
The Function Networking Environment

- **Default** - a default network environment within the VPC is provided for you
  - Access to the Internet always permitted to your function
  - No access to VPC-deployed assets

- **Customer VPC** - Your function executes within the context of your own VPC.
  - Privately communicate with other resources within your VPC.
  - Familiar configuration and behavior with:
    - Subnets
    - Elastic network interfaces (ENIs)
    - EC2 security groups
    - VPC route tables
    - NAT gateway
“Hold on…” – you (maybe)
Lots of existing ways to abstract away servers

- SaaS
- PaaS
- MBaaS
- *aaS
- Application engines/platforms
What’s unique about Lambda?

- Abstraction at the code/function level (arbitrary, flexible, familiar)
- The security model (IAM, VPC)
- The pricing model
- The community
- Integration with the AWS service ecosystem!
  - Scale
  - Triggers
Benefits of AWS Lambda for Building Serverless Applications

1) No Servers to Manage
AWS Lambda automatically runs your code without requiring you to provision or manage servers. Just write the code and upload it to Lambda.

2) Continuous Scaling
AWS Lambda automatically scales your application by running code in response to each trigger. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload.

3) Subsecond Metering
With AWS Lambda, you are charged for every 100ms your code executes and the number of times your code is triggered. You don't pay anything when your code isn't running.
Many Serverless Options on AWS

- Compute
- Storage
- Database
- Network
- Messaging and Queues
- Monitoring & Logging
- Content Delivery
- Security
- User Management
- Internet of Things
- Machine Learning
- Streaming Analytics
Example Serverless Architecture
PlayOn! Sports – Video stream processing

Laptop Encoders → HLS → CloudFront → S3 Ingest → Cascading Lambda Functions

- HQ Copy
- 480p Transcode
- 360p Transcode
- Audio-only Transcode
- Thumbnail
- QOS Analytics

→ CloudFront Streaming → S3 Playback → VOD Stream mobile client
→ Live stream mobile client
But...

... in order to utilize Lambda, do I really need to architect event-driven applications?

... is there a way I can use this construct to build multi-tier SOA applications?
The microservices approach still works.
Enter Amazon API Gateway

A managed service to…

- Create
- Configure
- Publish
- Maintain
- Monitor
- Secure

…your APIs!
Creating Your API

- Console, API, CLI support
- API REST
- Support for standard HTTP methods
- Swagger Import/Export
- Custom domains

Pet Store

/pets

/pets/{petId}
- GET
- POST
- PUT
Configuring Your API

- Choose what your APIs integrate with:
  - AWS service APIs (AWS Service Proxy)
  - Any other accessible web service
  - AWS Lambda!

- Add an optional managed cache layer
Publish Your API

API Configuration can be deployed to a stage

Stages are different environments

For example:

- Dev (e.g. awsapigateway.com/dev)
- Beta (e.g. awsapigateway.com/beta)
- Prod (e.g. awsapigateway.com/prod)
- As many stages as you need

You can use Mock Integrations
Maintain Your API

- Managed scaling
- Usage-based pricing
  - $3.50 per Million API requests received + Data Transfer Out
- Ability to create and require API keys for developer integration
- Generate client SDK programatically:
  - based on API deployments (Stages)
  - SDKs know how to handle throttling responses and sign requests with AWS temporary credentials (SigV4)
  - Support for Android, iOS, JavaScript, …
Monitor and Secure Your API

- Native CloudWatch metrics and CloudWatch Logs integration
- CloudTrail integration to track changes to your API
- Native integration with IAM and AWS Sigv4 to authorize access to APIs
- Custom authorization
- Integration with Amazon CloudFront for DDoS protection
Serverless Architecture Patterns
Microservices
Web Applications

Example: Weather Application

- S3
  - Weather app hosted in S3

- User looks up local weather information

- API GATEWAY
  - App makes REST API call to endpoint
  - Lambda is triggered

- Lambda runs code to retrieve local weather information and returns data back to user

- DYNAMO DB
Mobile Backend

Example: Mobile Backend for Social Media App

User posts status update

API GATEWAY

App makes REST API call to endpoint

Lambda is triggered

Lambda runs code to look up friends list and pushes status update notification to user’s friends

SNS
Real-time Analytics Engine

Example: Analysis of Streaming Social Media Data

Lambda is triggered

KINESIS
Social media stream is loaded into Kinesis in real-time.

Lambda runs code that generates hashtag trend data and stores it in DynamoDB

DYNAMODB

Social media trend data immediately available for business users to query
Serverless Best Practices
AWS Lambda Best Practices

- Limit your function size – especially for Java (starting the JVM takes time).
- Don’t assume function container reuse – but take advantage of it when it does occur.
- Don’t forget about disk (500 MB /tmp directory provided to each function).
- Node – remember execution is asynchronous.
- Use the included logger (include details from service-provided context).
- Create custom metrics (operations-centric, and business-centric).
Amazon API Gateway Best Practices

- Use Mock integrations
- Combine with Cognito for managed end user-based access control.
- Use stage variables (inject API config values into Lambda functions for logging, behavior)
- Take ownership of HTTP response codes
- Use Swagger import/export for cross-account sharing
Additional Best Practices

- Use strategic, consumable naming conventions (Lambda function names, IAM roles, API names, API stage names, etc.)
- Use naming conventions and versioning to create automation.
- Externalize authorization to IAM roles whenever possible
- Least privilege and separate IAM roles
- Externalize configuration – DynamoDB is great for this.
- Contact AWS Support before known large scaling events
- Be aware of service throttling, engage AWS support if so.
Serverless Reference Architectures

https://github.com/awslabs
Search for lambda-refarch
Reference architecture: Web applications with AWS Lambda

https://github.com/awslabs/lambda-refarch-webapp
Reference architecture: Mobile backend using AWS Lambda and Amazon API Gateway

https://github.com/awslabs/lambda-refarch-mobilebackend
Reference architecture: Real-time stream processing using AWS Lambda and Amazon Kinesis

https://github.com/awslabs/lambda-refarch-streamprocessing
Reference architecture: IoT backend using AWS Lambda and Amazon Kinesis

https://github.com/awslabs/lambda-refarch-iotbackend
Reference Architecture: Real-time File Processing using AWS Lambda

https://github.com/awslabs/lambda-refarch-fileprocessing

AWS LAMBDA: REAL-TIME FILE PROCESSING

Building real-time file processing systems can be complex and expensive. Lambda architecture can result in low utilization rates of compute infrastruc-
ture. AWS Lambda provides a proven experience that can help you build
real-time processing applications in parallel without the need to provi-
sion or manage servers. Lambda can also provide application processing by running
code in parallel in response to triggers from AWS services such as Amazon
S3.
Let’s Build Something Serverless…

- Try the 45 AWS Lambda blueprints!
- Use the Reference Architecture CloudFormation templates
Thank you!