Building Event-Driven Serverless Applications

Danilo Poccia, Technical Evangelist

@danilop  danilop
The Monolithic Architecture
The Service-Oriented Architecture
The N-Tier Web Application

Presentation Tier

Logic Tier

Data Tier
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 should my app withstand a server failing?
- Which users should have access to my servers?
- Which OS should my servers run?
- How can I tell if a server has been compromised?
- How will the application handle server hardware failure?
- How can I increase utilization of my servers?
- Which packages should be baked into my server images?
- How should I implement dynamic configuration changes on my servers?
- When should I decide to scale out my servers?
- 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 budget?
- How much load for my servers?
- How will the application handle server hardware failure?
- Which users should have access to my servers?
- Should I tune OS settings to optimize my application?
- How many users create too much load for my servers?
- How many servers should I budget for?
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
Enter Amazon API Gateway

A managed service to...

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

...your APIs!
<demo>

...
</demo>
Serverless Architecture Patterns
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

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

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

Lambda is triggered

DYNAMODB
Lambda runs code that generates hashtag trend data and stores it in 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)
- Node – remember execution is asynchronous.
- 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)
- 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)
- Use request/response mapping templates everywhere within reason, not passthrough.
- 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.
Money is a renewable resource, 
Time is not
Thank you!