Java Champions

Getting started with Serverless Java

jChampions Conference

January 2022

Hello!

I am Alan Williamson

Java Champion ▪ Book Author ▪ Speaker ▪ Podcaster ▪ CTO

Upcoming book; **“Think like a CTO”** Manning Publications

You can find me at https://alan.is/

The problem with Java, is not the code, but all the fluff we have to manage before our code is executed.

What if we could get rid of the fluff?

Dream with me for a minute

Servlet

Imagine writing a servlet without the logistics of the application server

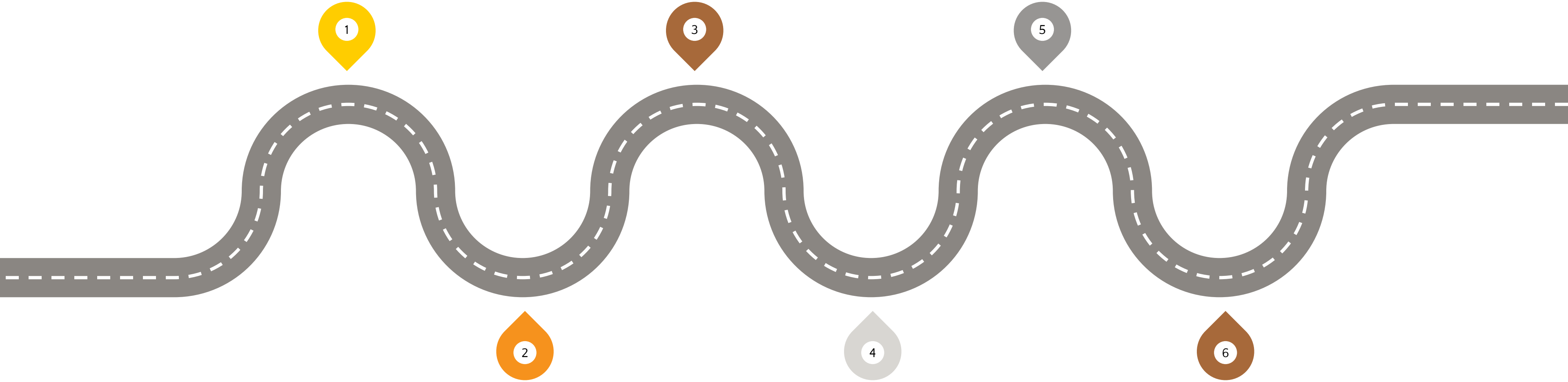
JMS

Imagine writing a piece of code to process jobs, off a queue, without worrying about hosting or scaling

Timer

Imagine writing a piece of code that gets triggered on a timer without wrestling with cron script, timer thread or application server setup

What we are going to cover



1 Serverless Introduction

2 Autonomy of a function

3 Live Coding Writing/Deploying

4 Lambda everywhere!

5 Lambda best practices

6 Useful Links

1 Serverless Introduction

Serverless IS NOT Serverless

Of course there is a server – let us not pretend otherwise – *instead*

Serverless is free of any underlying server management

So instead of “Serverless” think

- **Function-as-a-Service (FaaS)**

The ability to focus on the job at hand, and not be concerned about the logistics of initialization, scalability, configuration or performance.

Function-as-a-Service

Deploy functions; not apps

Only deploy what is needed to service that event, not the whole application

Scalability

Automatically scales up/down depending on the incoming traffic (limits can imposed)

Per Invocation Billing

Charged only by the function call (tiered on the memory provisioned)

Zero Cost At Rest

When there is no traffic, there is no cost being incurred.

Faster Release / Zero downtime

Only deploy/update the function not the whole enterprise, with zero downtime release

Thorough Testing

Greater confidence and code coverage for testing, as you are testing only small functions not a complete app each time

Logistically how does this work?

1 Implement a method

As if you were writing a servlet; you create a class and implement a method from an interface

2 Package the function

Like creating a WAR, compile the code into a single JAR file, that is packaged up, with some configuration parameters such as the language, memory, logging, security considerations

3 Deploy the function

Push the file to the cloud, which will then unpack it and make it ready for execution once an event is triggered

2

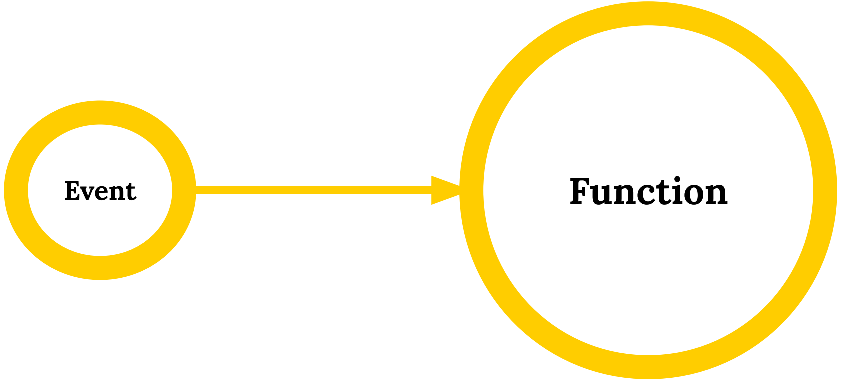
Autonomy of a Function

AWS Lambda

AWS Lambda != Java Lambda

Did we learn nothing from the whole Java / Javascript naming debacle??

Event based Processing



- Functions are executed as a result of an event (*such as an HTTP request*)
- A function is provisioned to use a maximum amount of memory
- They only run for a specific amount of time before being terminated (*for example 30 seconds to service an API Gateway HTTP event; 15 mins otherwise*)
- Designed to run in a parallel to cope with dynamic load

How much does it cost?

requests + runtime = cost

$0.20 + **$0.0000166667**

Per 1 million requests

Per GB-second

Runtime cost; for each 1ms running with 128MB memory it would be:

$0.0000000021

https://aws.amazon.com/lambda/pricing/

AWS Pricing Illustrations ($ per month)

Memory	256 MB			512 MB		
Duration	50 ms	100 ms	250 ms	50 ms	100 ms	250 ms
1,000 per day	$0	$0	$0	$0	$0	$0
10,000 per day	$0	$0	$0	$0	$0	$0
1,000,000 per day	$5.88	$5.88	$15.06	$11.89	$24.56	$62.58
1,000 per hour	$0	$0	$0	$0	$0	$0
10,000 per hour	$1.26	$1.26	$1.26	$1.26	$1.26	$9.80
1,000,000 per hour (277 per second)	$291.22	$443.30	$899.55	$443.30	$747.47	$1659.97

Source: https://calculator.aws/#/createCalculator/Lambda

There are some ~~limitations~~ guidelines

Memory Size

Provision from 128 MB to 10 GB, in 1-MB increments.

Execution Time

Maximum time 15 minutes; though API Gateway is 30 seconds

Temporary Disk Space

512MB disk storage in /tmp/ for use; can persist between requests; but don't rely on it

Deployment Package Size

50MB zipped; though can use bigger deployments via layers (common library code)

Concurrency

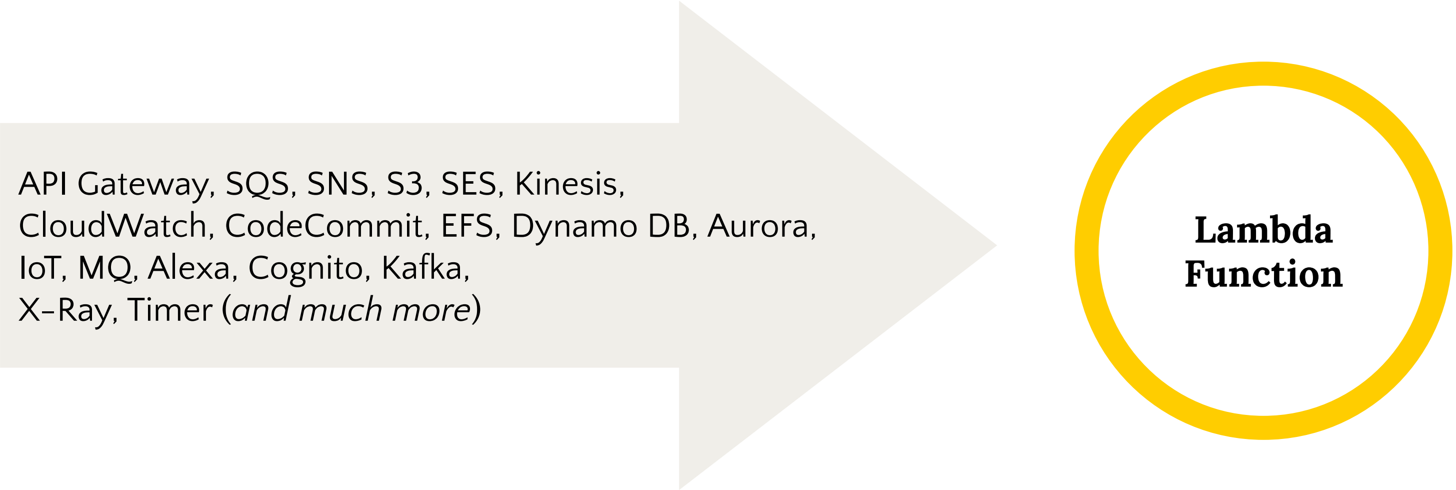
Default to 1,000; but can be increased through a support request to 10,000+

3

Writing/Deploying Lambda

```
boolean liveCoding = true;
```

4 Lambda is everywhere



API Gateway, SQS, SNS, S3, SES, Kinesis,
CloudWatch, CodeCommit, EFS, Dynamo DB, Aurora,
IoT, MQ, Alexa, Cognito, Kafka,
X-Ray, Timer (*and much more*)

Lambda Function

… and the basic principle is all the same; create a function to process an event

5 Best Practices

Keep Small

Utilize as little memory as possible, and keep packages small to decrease startup time

Warm up your Lambda

Lambda can go cold; this is when your Lambda is removed from an execution ready state

Configuration in SSM

Put configuration in SSM and cache them.

Do not use Threads

Threads are paused after function executes. Instead think of your Lambda as single thread, with multiple running ones being your threads

Avoid Connection Pooling

Open up connection to database when you need it; do not cache it between function calls. This will make code more tolerant

Don’t do too much

Don’t have your lambda’s doing too much; instead chain them together, or use a queue, to keep them lean and fast

Long Running Lambda

Technique #1 (old school)

Split up your long running job into smaller jobs, and use SQS to trigger the execution for a maximum of 15 minutes, before putting another job on the queue to take over

Technique #2 (preferred)

Fargate is a serverless micro container service for spinning up, on-demand, containers to process long running jobs. Charged on a per-second basis.

Think of it like firing up a JVM to run a program without having to worry about servers. Sound familiar?

Lambda != WAR/EAR

!! Warning !!

A Lambda package is not a WAR/EAR file; it does not contain web content, or any other JAR/WAR files.

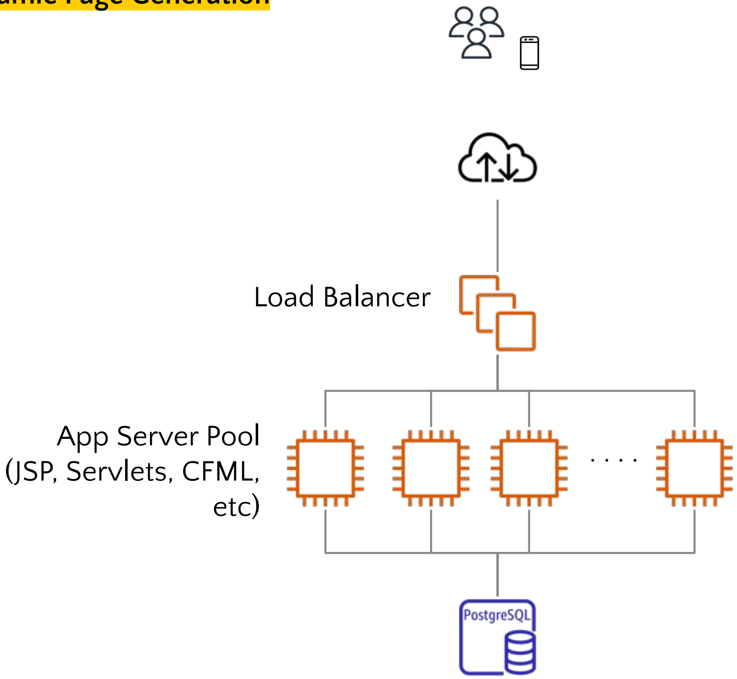
Instead of JSP, think static-web-sites served from a CDN/S3 utilizing JavaScript (React/Angular/JQuery) to create dynamic content via AJAX.

Extremely scalable and fault tolerant system

Trust me – your web developers will welcome the freedom

Typical Dynamic Website Pattern

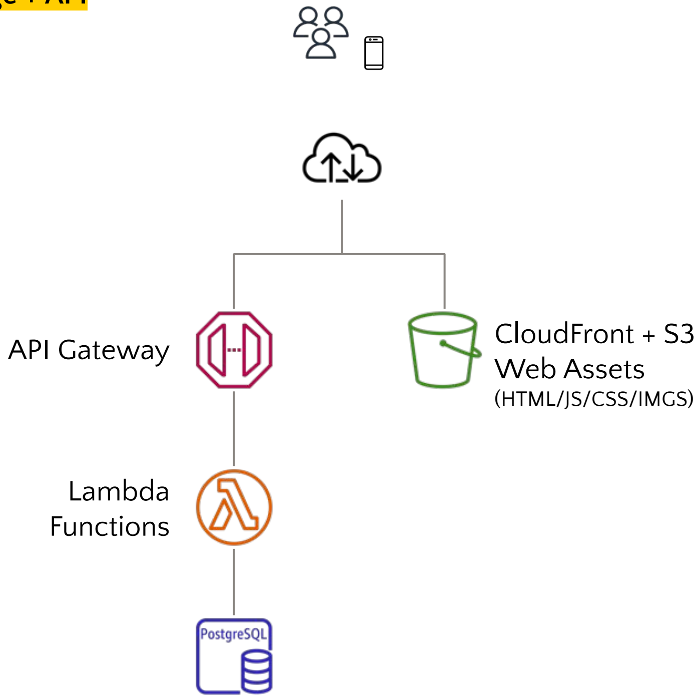
Dynamic Page Generation



Load Balancer

App Server Pool
(JSP, Servlets, CFML,
etc)

Static Page + API



API Gateway

CloudFront + S3
Web Assets
(HTML/JS/CSS/IMGS)

Lambda
Functions

~~The problem with Java, is not the code, but all the fluff we have to manage before our code is executed.~~

Serverless Java removes all the ‘container’ logistics and lets us scale up and down automagically

This is what the App Server should have been from the start

6 Further Reading

- https://aws.amazon.com/lambda/
- https://aws.amazon.com/api-gateway/pricing/
- https://docs.aws.amazon.com/lambda/latest/dg/lambda-java.html
- https://aws.amazon.com/fargate/
- https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html
- https://docs.aws.amazon.com/lambda/latest/dg/lambda-services.html
- https://www.serverless.com/framework/docs/getting-started
- https://www.bschaatsbergen.com/behind-the-scenes-lambda

Thank you

Any questions ?

Find me at

- https://www.linkedin.com/in/a1anw2/
- https://alan.is/ alan@alan.is
- https://github.com/a1anw2/jchampionconf2022