

# Hálózatba kapcsolt erőforrás platformok és alkalmazásaik

Simon Csaba

TMIT, HSNLab

2019

# Microservices

- An architecture pattern decomposing vertically a monolith system into loosely coupled subsystems (microservices). Nothing more. The pattern doesn't dictate how it should be done technically.
- Just to highlight it: Microservices and Containers are **not the same** and **totally independent**. You can use Docker with a monolith app, and you can have several micro-services without using Docker at all.

# Architecture pattern evolution

- **Monolith**
  - distributed horizontally on Prem/Cloud/Hybrid
  - containerized horizontally
- **Microservices**
  - distributed vertically and horizontally on Prem/Cloud/Hybrid
  - containerized vertically and horizontally

# Serverless features

- A unit of work consumes resources only when it is used
  - Function is a unit of work
    - stateless, short lived
    - serves one goal
    - arguments (input) and result (output)
- Orchestration of independent pieces of work (functions as a service FaaS)
  - Carrying state of the entire flow (program)
  - Error handling
  - Transaction management

# Serverless model

- Focus on business logic
- Code centric paradigm. Hyde Infrastructure.
  - Focus on coding resolving business problems and forget about infrastructure
  - Everything is working on some “computing resources”
- Scalability
  - Developers don't do anything for scaling. Scaling is handled automatically.
- Billing
  - Don't pay for idle time
  - Pay for milliseconds
- Utilization

# Serverless Platform

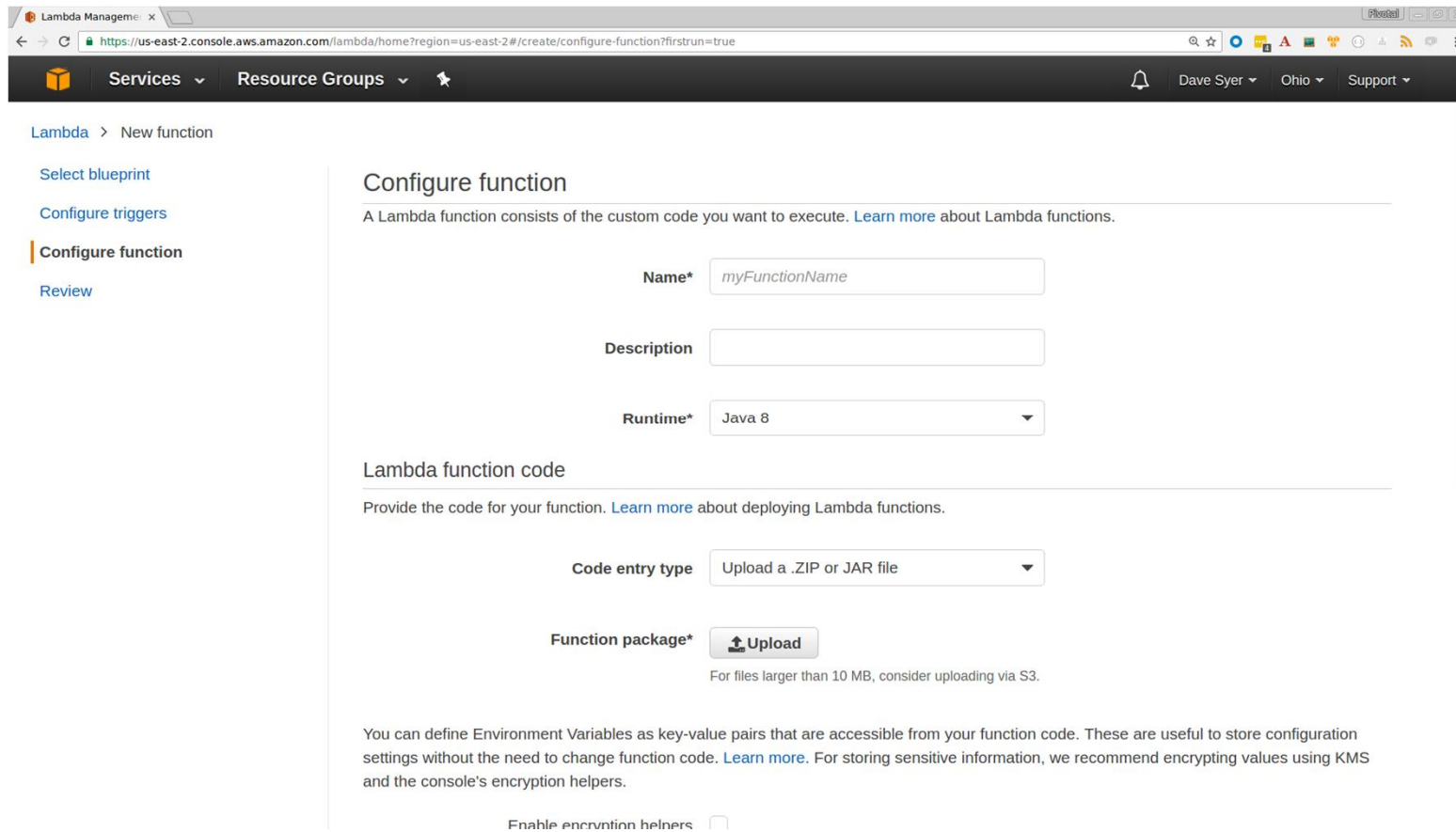
- Public clouds:
  - AWS Lambda
  - Google Cloud Functions
  - Azure Functions
  - IBM (OpenWhisk based)
  - Oracle (fn based)
- Open source frameworks:
  - OpenWhisk
  - Kubeless
  - Fn
  - OpenFaaS
  - Fission
  - ...and many more



Multiple language support (availability depends on the framework):

- NodeJS
- Python
- Java
- Scala
- Clojure
- ...

# Amazon Lambda



The screenshot shows the AWS Lambda console interface for configuring a new function. The browser address bar shows the URL: `https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/create/configure-function?firstrun=true`. The navigation bar includes 'Services' and 'Resource Groups' menus, and user information for 'Dave Syer' in 'Ohio' with a 'Support' link. The left sidebar contains a breadcrumb 'Lambda > New function' and a list of steps: 'Select blueprint', 'Configure triggers', 'Configure function' (which is highlighted), and 'Review'. The main content area is titled 'Configure function' and includes a sub-header 'Configure function' and a descriptive sentence: 'A Lambda function consists of the custom code you want to execute. [Learn more](#) about Lambda functions.' Below this are three input fields: 'Name\*' with the value 'myFunctionName', 'Description' (empty), and 'Runtime\*' with a dropdown menu set to 'Java 8'. The next section is 'Lambda function code', with a sub-header 'Lambda function code' and a descriptive sentence: 'Provide the code for your function. [Learn more](#) about deploying Lambda functions.' It features a 'Code entry type' dropdown menu set to 'Upload a .ZIP or JAR file' and a 'Function package\*' section with an 'Upload' button. A note below the button states: 'For files larger than 10 MB, consider uploading via S3.' At the bottom, there is a paragraph about Environment Variables: 'You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#). For storing sensitive information, we recommend encrypting values using KMS and the console's encryption helpers.' Below this paragraph is a checkbox labeled 'Enable encryption helpers' which is currently unchecked.

# AWS Lambda

- One of the first Function projects on the market
  - The idea is:
    - You have your code written with one of supported languages (limited list, binaries are preconfigured by AWS)
    - Your code exposes some standardized API
    - You upload your code (in a zip file) to AWS Lambda
    - Basing on event (e.g. request on URL) AWS Lambda allocates resources for your code, invokes a function and releases the resource at the end
    - The flow is orchestrated with AWS Step Functions
    - Visual Flow designer



# Google Cloud Function

The screenshot shows the Google Cloud Platform console interface. At the top, there's a navigation bar with the Google Cloud Platform logo and the project name 'CF-Spring-funkytown'. Below this, the 'Cloud Functions' section is active, displaying 'Function details' for 'helloWorldJava'. The 'Source' tab is selected, showing a code preview for 'Hello.java'. The code is a simple Java class with a main method that prints 'Hello ' followed by the first argument. Below the code, the 'Source location' is listed as 'cfsf-hello-world-java/us-central1-helloWorldJava-qpllmznutury.zip' and the 'Function to execute' is 'helloWorldJava'.

Cloud Functions - CF x

https://console.cloud.google.com/functions/details/us-central1/helloWorldJava?project=cf-spring-funkytown&tab=source&duration=PT1H

Sign up for a free trial and you'll get \$300 in credit and 12 months to explore all of Google Cloud Platform. [Learn more](#) Contact sales: 0800 026 1212 (9am - 5pm GMT) [DISMISS](#) [SIGN UP FOR FREE TRIAL](#)

Google Cloud Platform CF-Spring-funkytown

Cloud Functions < Function details EDIT DELETE COPY VIEW LOGS

helloWorldJava

General Trigger **Source** Testing

Code preview

Hello.class **Hello.java** index.js package.json

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello " + args[0]);
    }
}
```

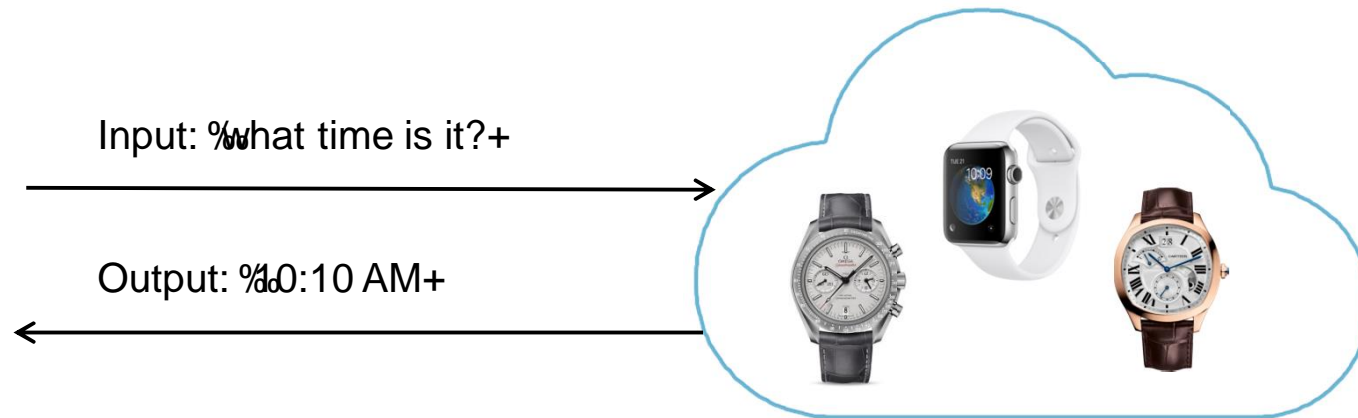
Source location

[cfsf-hello-world-java/us-central1-helloWorldJava-qpllmznutury.zip](#)

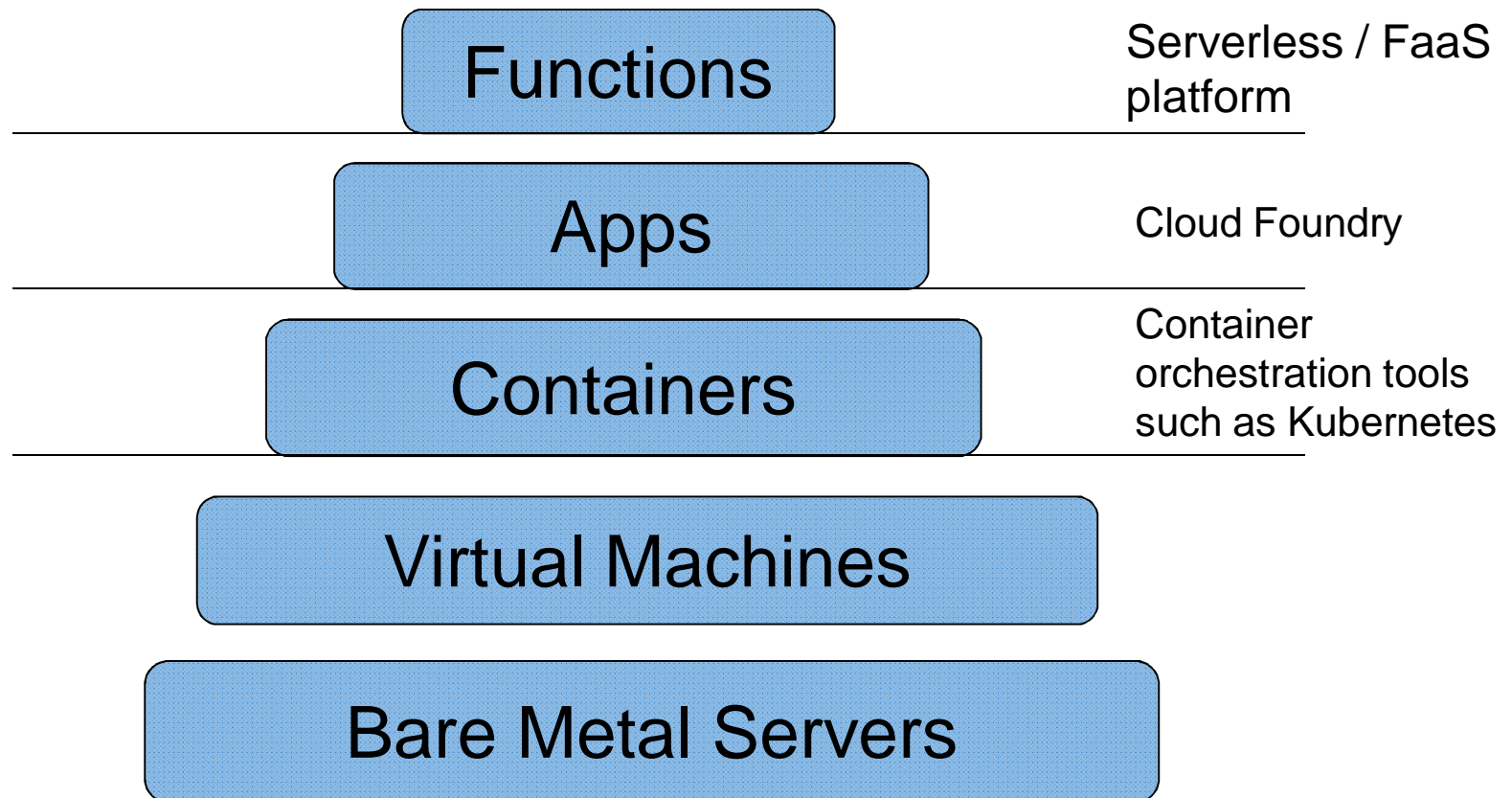
Function to execute

helloWorldJava

# What is a function?



# Cloud Abstractions



# Function = the user code sent to a serverless platform

- Static self-running piece-of-work wrapped into a container with everything it needs for its work
  - code + platform
  - stateless
  - single purposed
  - arguments (input) and result (output)

# Challenges of Serverless / FaaS

- New architectural style
- Management of large populations of functions
- Vendor lock-in
- Execution duration limit
- Start up latency
- Network latency among functions
- Immature tooling for development
- Immature tooling for Day 2