

# Felhő alapú hálózatok

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 displays the URL: `https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/create/configure-function?firstrun=true`. The top navigation bar includes 'Services' and 'Resource Groups' menus, and user information for 'Dave Syer' in the 'Ohio' region. 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 the following sections:

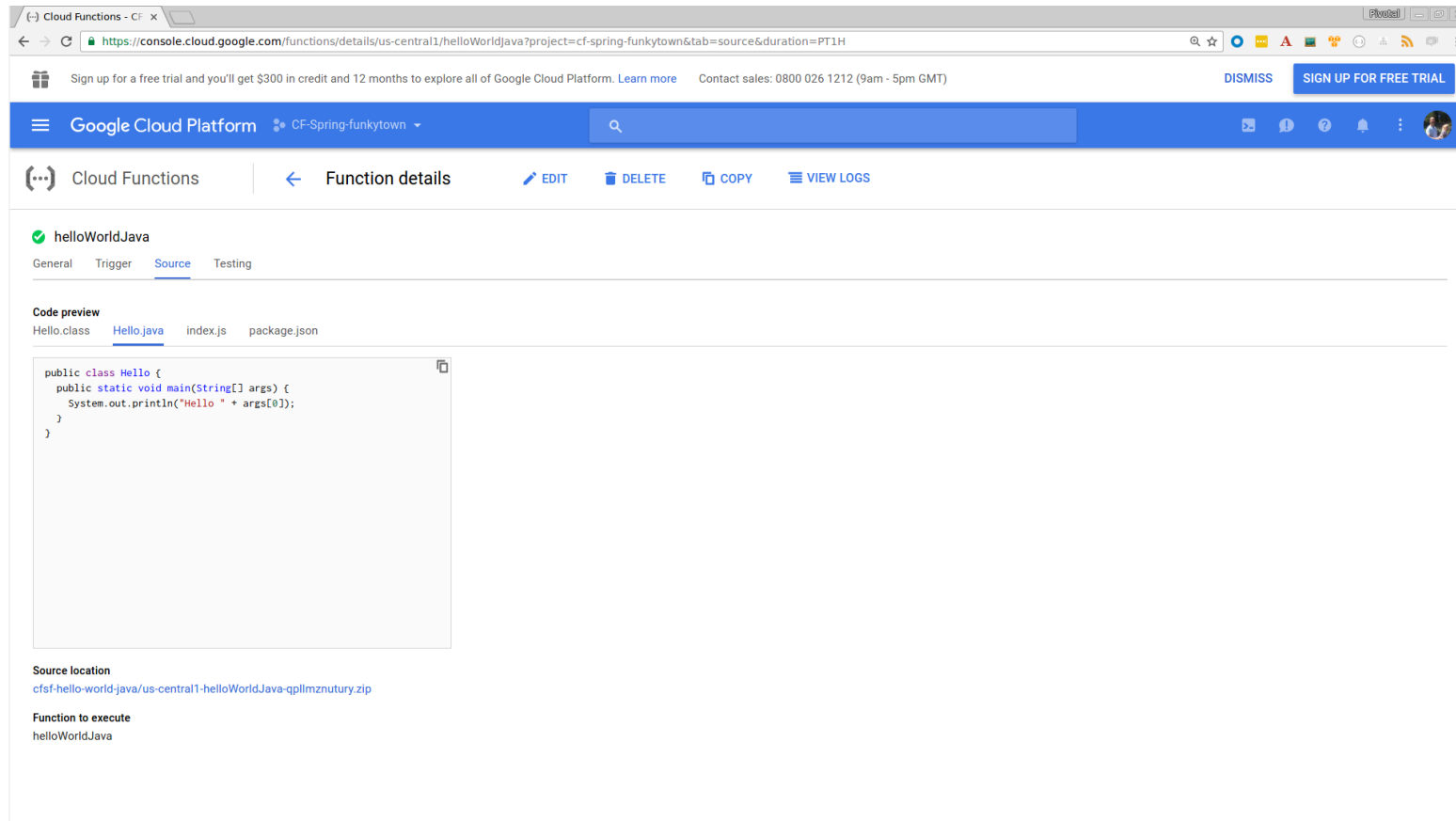
- Configure function**: A sub-header with a note: 'A Lambda function consists of the custom code you want to execute. [Learn more](#) about Lambda functions.'
- Form fields**:
  - Name\***: A text input field containing the placeholder text 'myFunctionName'.
  - Description**: An empty text input field.
  - Runtime\***: A dropdown menu currently set to 'Java 8'.
- Lambda function code**: A sub-header with a note: 'Provide the code for your function. [Learn more](#) about deploying Lambda functions.'
- Code entry type**: A dropdown menu set to 'Upload a .ZIP or JAR file'.
- Function package\***: A button labeled 'Upload' with an upward arrow icon. Below it, a note reads: 'For files larger than 10 MB, consider uploading via S3.'
- Environment Variables**: A paragraph explaining that environment variables are key-value pairs accessible from the function code, useful for configuration. It includes a link to 'Learn more' and a recommendation to use KMS for sensitive information. Below this text 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

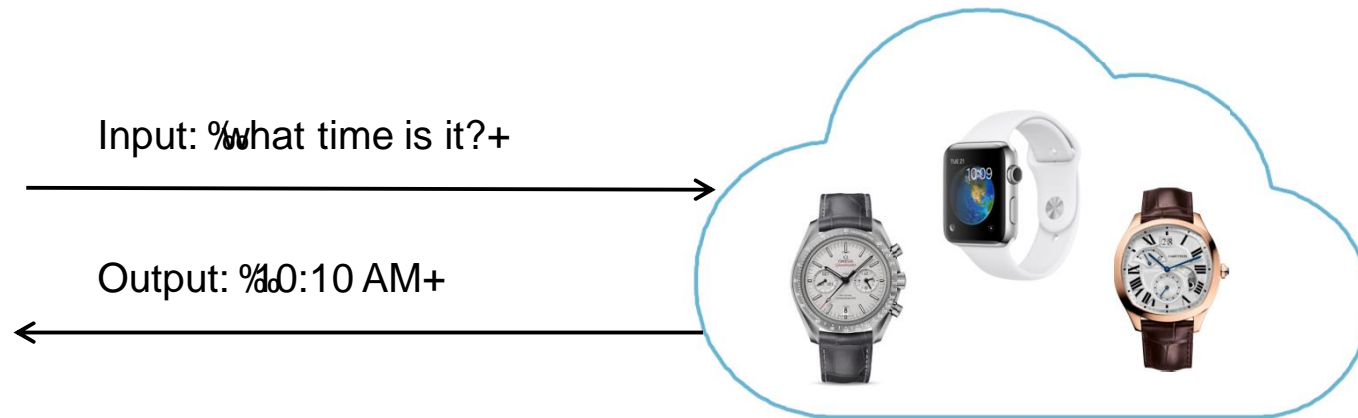


The screenshot displays the Google Cloud Platform console interface. At the top, there's a navigation bar with the Google Cloud Platform logo and the project name 'CF-Spring-funkytown'. Below this, the 'Cloud Functions' section is active, showing the 'Function details' for 'helloWorldJava'. The 'Source' tab is selected, displaying the following Java code:

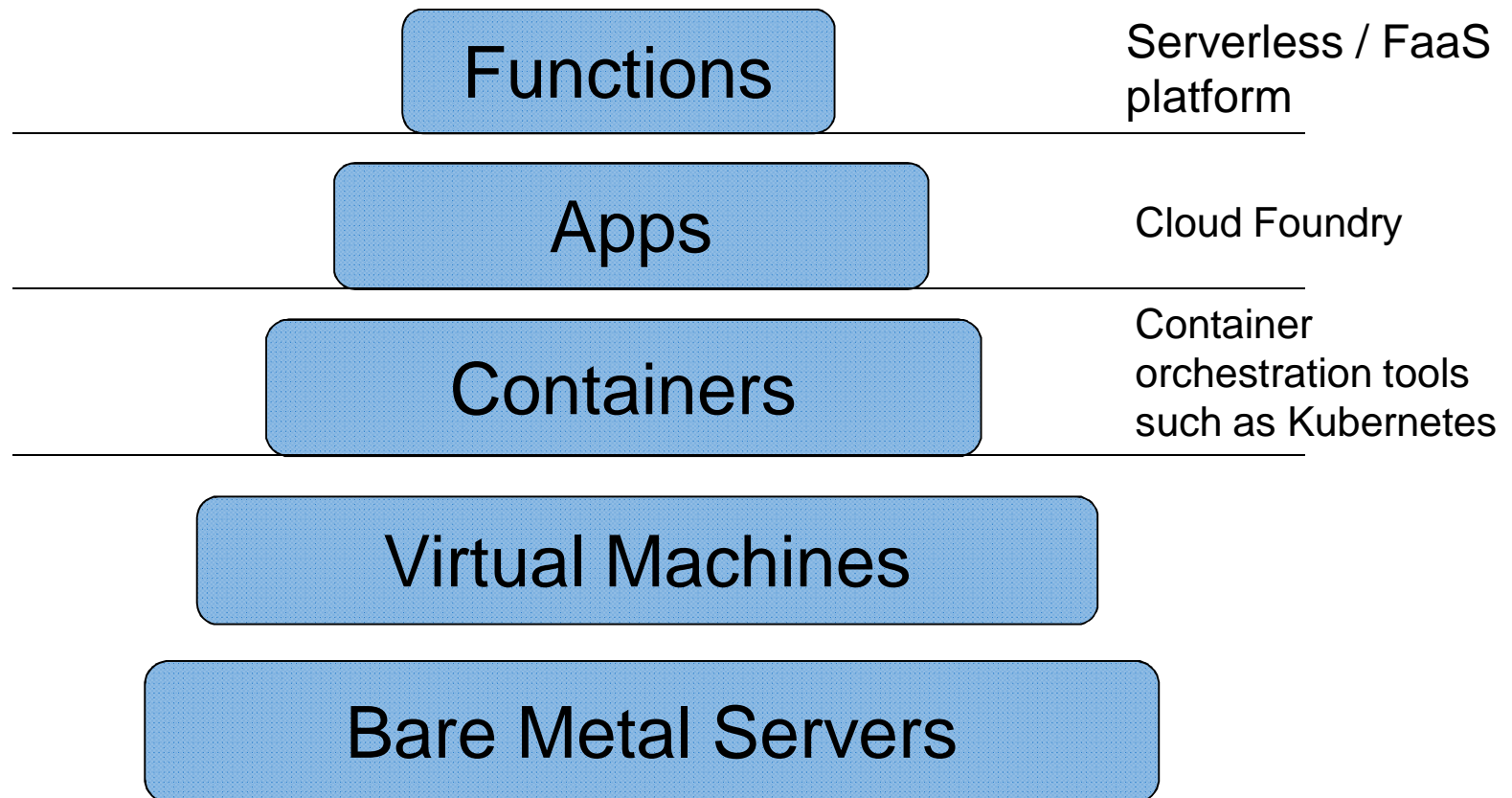
```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello " + args[0]);
    }
}
```

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'.

# 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