



adaptTo()

APACHE SLING & FRIENDS TECH MEETUP
2 - 4 SEPTEMBER 2019



Sling & Serverless - Best Friends Forever?

Bertrand Delacrétaz - Principal Scientist, Adobe

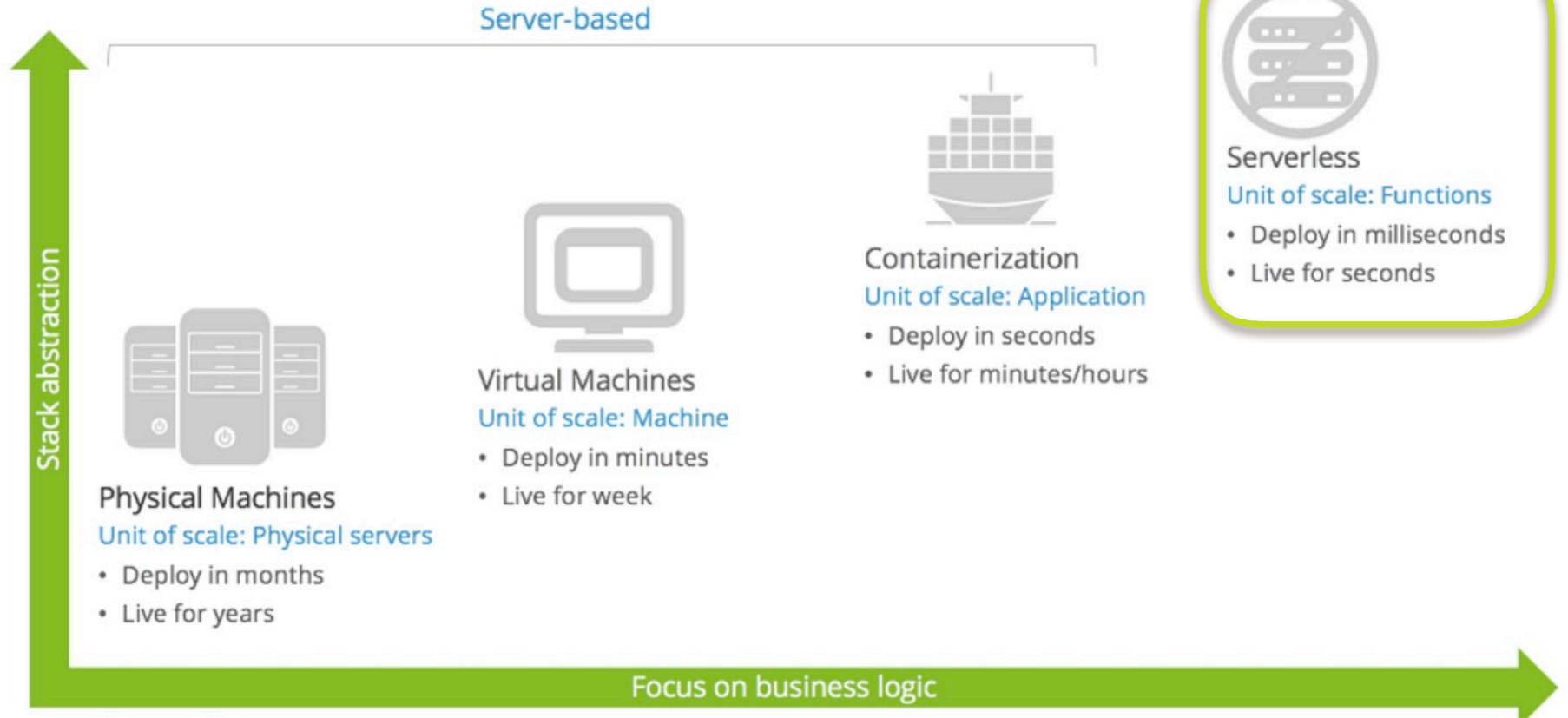


Images: stock.adobe.com, unless otherwise specified
slides revision: 2019-09-03

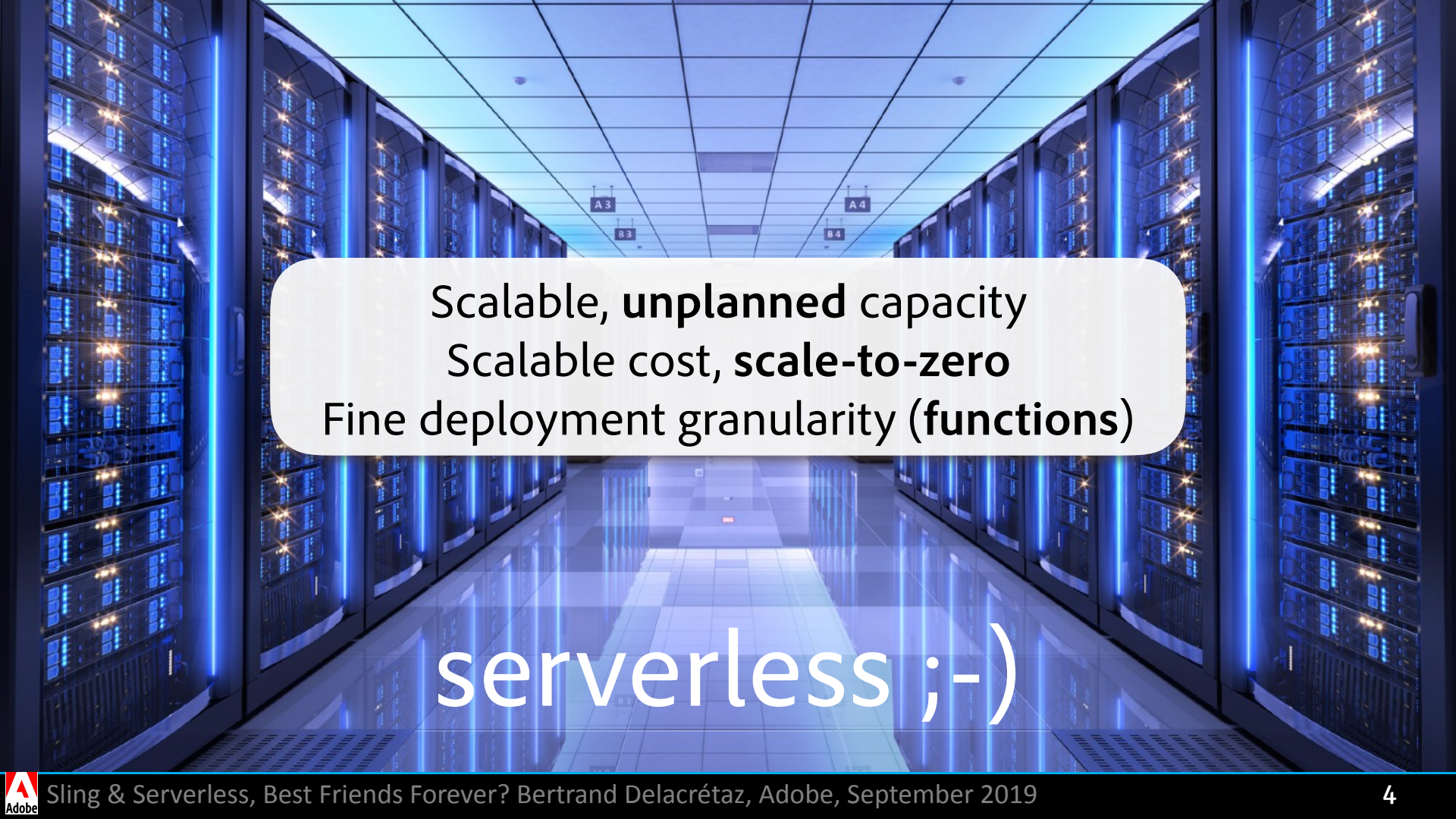


Serverless?

Cloud Technologies Evolution



Source: Deloitte Consulting LLP, via Alexander Klimetschek, @alexkli



Scalable, **unplanned** capacity
Scalable cost, **scale-to-zero**
Fine deployment granularity (**functions**)

serverless ;-)

Hello, Serverless World!

```
// The actual OpenWhisk action code
```

```
function main(params) {  
  const name = params.name || 'World';  
  
  const content = `  
    <html>  
      <body>  
        <h1>Hello, ${escapeForHTML(name)}!</h1>  
      </body>  
    </html>  
  `;  
  
  console.log(content);  
  return {body: content };  
}
```

Installation:

```
wsk action update web-hello web-hello.js --web true
```

URL:

```
wsk -i action get web-hello --url
```





Units of Deployment

Serverless: Containers or Functions ?

Scalable, unplanned capacity + Scalable cost, scale-to-zero



```
// The actual OpenWhisk action code
function main(params) {
  const name = params.name || 'World';

  const content = `
    <html>
      <body>
        <h1>Hello, ${escapeForHTML(name)}!</h1>
      </body>
    </html>
  `;

  console.log(content);
  return {body: content };
}
```

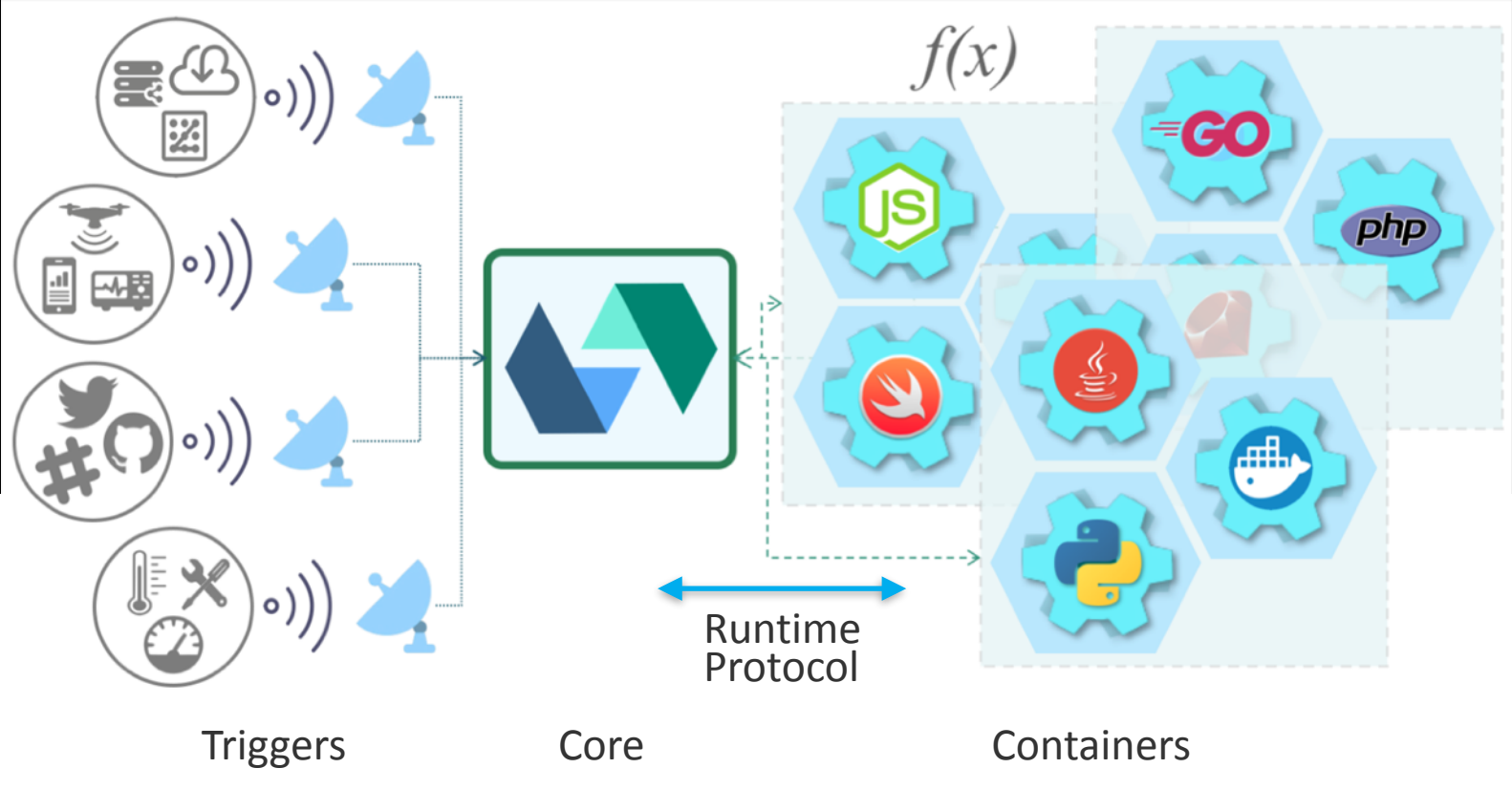
Fine deployment granularity ?

Needs fast startup

OpenWhisk: simple init/run HTTP API

Limited choice of languages?

OpenWhisk: Orchestrated Containers



OpenWhisk runtime protocol

The /init end point

After starting the Docker container, the invoker will POST to /init the following payload:

```
{
  "value": {
    "name" : "helloPHP",
    "main" : "main",
    "binary": false,
    "code" : "<?php ...",
  }
}
```

The /run end point

To run the action, the invoker will POST to /run. This may happen multiple times, but never concurrently. The payload contains the arguments to be passed to the function to be executed. For example:

```
{
  "value":
  {
    "name" : "Rob",
  }
}
```

Runtime container responds to **/init** and **/run** HTTP requests
-> easy to create **any** runtime



GraalVM - native Java

(in containers?)

graalvm native fast startup

Tous Actualités Maps Images Vidéos Plus Paramètres Outils

Environ 35900 résultats (0,39 secondes)

Speed up application launch time with GraalVM - De Bijenkorf ...
<https://medium.com/.../speed-up-application-launch-time-with-graa...> Traduire cette page
6 déc. 2018 - Startup time of your application is crucial to allow fast scaling. ... SubstrateVM (part of GraalVM) is a native virtual machine that allows you to ...

Instant Netty Startup using GraalVM Native Image Generation - Medium
<https://medium.com/graalvm/instant-netty-startup-using-graalvm-n-...> Traduire cette page
22 mai 2018 - Instant Netty Startup using GraalVM Native Image Generation allows us to skip class initialization at run time, which is crucial for fast startup.

Helidon flies faster with GraalVM | Dmitry's Technical Blog
<https://dmitrykornilov.net/2019/.../helidon-flies-faster-with-graalv-...> Traduire cette page
17 avr. 2019 - GraalVM is an open source, high-performance, polyglot virtual machine ... A native executable offers important benefits, like shorter startup time ...

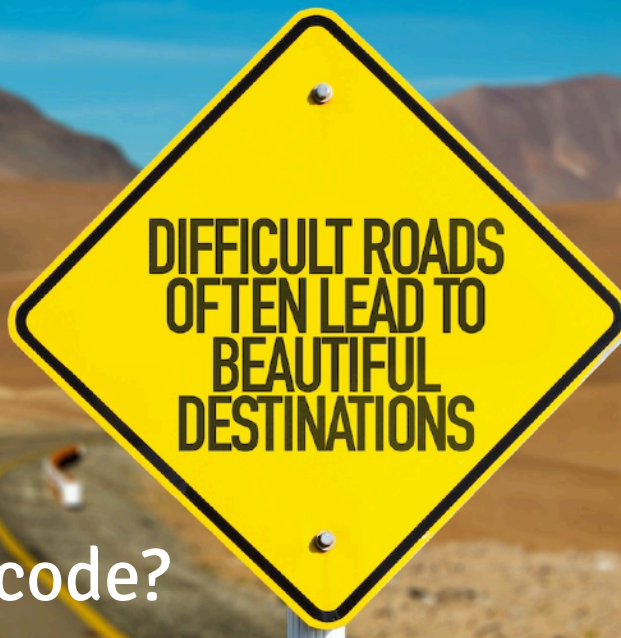
Why is GraalVM so fast? - Beginners - ClojureVerse
<https://clojureverse.org/CommunityCenter/Beginners> Traduire cette page
Kept seeing updates from GraalVM and my friends talked about that for several times. ... and JVM has been optimized for so many years, By fast I'm mean startup speed. ... What is fast is native-image compilation it offers for Java programs.

Why GraalVM
<https://www.graalvm.org/docs/why-graal/> Traduire cette page
Run Java Faster. GraalVM can run in the context of ... Running your application inside a Java VM comes with startup and footprint costs. GraalVM has a feature to create native images for existing JVM-based applications. The image generation ...
For Java Programs · For Node.js Programs · For Ruby, R, or Python

Native Image Example - GraalVM

Impressive startup speed...assuming you pass the native-image tool hurdle!

SubstrateVM limitations



OSGi ?

Legacy code?

What	Support Status
Dynamic Class Loading / Unloading	Not supported
Reflection	Supported (Requires Configuration)
Dynamic Proxy	Supported (Requires Configuration)
Java Native Interface (JNI)	Mostly supported
Unsafe Memory Access	Mostly supported
Class Initializers	Supported
InvokeDynamic Bytecode and Method Handles	Not supported
Lambda Expressions	Supported
Synchronized, wait, and notify	Supported
Finalizers	Not supported
References	Mostly supported
Threads	Supported
Identity Hash Code	Supported
Security Manager	Not supported
JVMTI, JMX, other native VM interfaces	Not supported
JCA Security Services	Supported

<https://github.com/oracle/graal/blob/master/substratevm/LIMITATIONS.md>

Sling code -> native code experiments

```
/** This is where we wire the system, like the OSGi framework
 * would do. As it seems hard to run that framework in a GraalVM
 * environment for now, we wire things statically.
 */
private static OsgiContext initialize() {
    final OsgiContext result = new OsgiContext();

    // This would be automatic in a JUnit environment
    result.registerInjectActivateService(new MockEventAdmin());

    // Our minimal resource provider
    final MockResourceProvider mrp = new MockResourceProvider();
    result.registerInjectActivateService(mrp);

    // SlingRequestProcessor
    result.registerInjectActivateService(new SlingRequestProcessorWrapper(result.bundleContext()));

    // ResourceResolver
    //result.registerInjectActivateService(new MockResourceResolver(mrp));
    result.registerInjectActivateService(new MockServiceUserMapper());
    result.registerInjectActivateService(new ResourceAccessSecurityTracker());
    final ResourceResolverFactoryActivator rrfaf = new ResourceResolverFactoryActivator();
    result.registerInjectActivateService(rrfaf);
    result.registerInjectActivateService(new ResourceResolverFactoryService(rrfaf));

    return result;
}
```

Static wiring with
Sling OSGi mocks

Static wiring worked well
ok (with lots of suffering)
until hitting what's
probably a native-image
bug exposed by our legacy
code.

See [SLING-8556](#) and [sling-whiteboard/graalvm](#) - and
with more effort or once the
tools mature that might
work.



I GIVE UP!

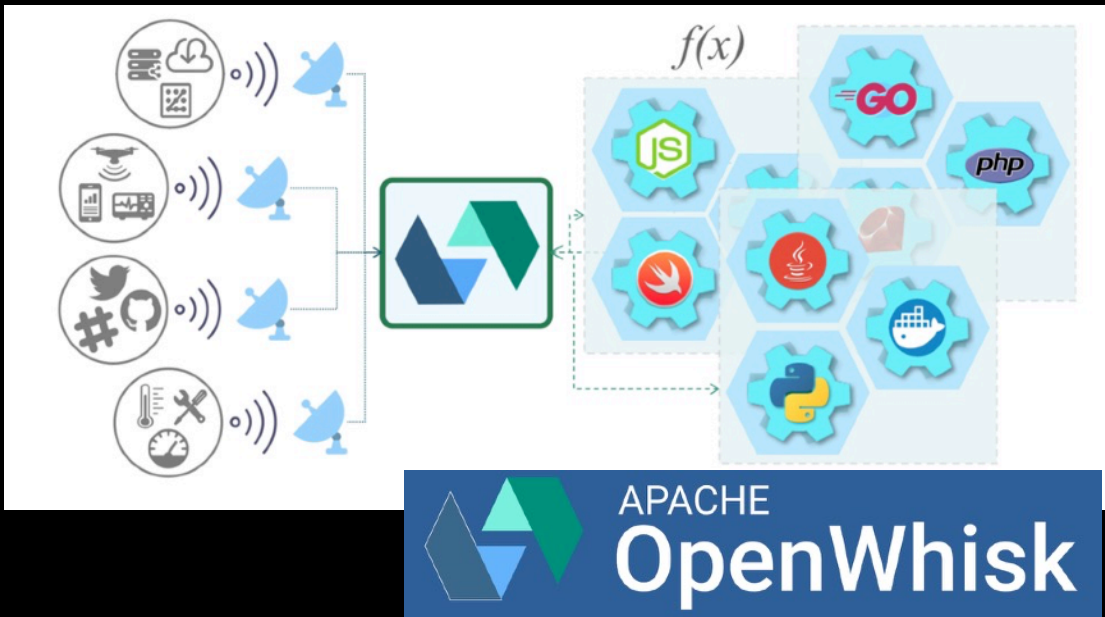


Embracing the Platform?



**Embrace
The Platform!**

serverless == JavaScript?



Are we doing Google Search Driven Architecture now?. Hmm...

The screenshot shows a Google search results page for the query "serverless javascript". The search bar at the top contains the text "serverless javascript" and a magnifying glass icon. Below the search bar, there are navigation links for "Tous", "Images", "Actualités", "Vidéos", "Maps", "Plus", "Paramètres", and "Outils". The search results are displayed in French. The first result is "serverless/serverless: Serverless Framework – Build web ... - GitHub" with a link to <https://github.com/serverless/serverless>. The second result is "How Does Serverless JavaScript Work? Service Workers and ..." with a link to <https://www.cloudflare.com/learning/serverless/serverless-javascript/>. The third result is "La puissance du JavaScript serverless | Udemy" with a link to <https://www.udemy.com/la-puissance-du-javascript-serverless/>. The fourth result is "Next.js 8 supporte désormais les applications serverless - Le Monde ..." with a link to <https://www.lemondeinformatique.fr/.../lire-nextjs-8-supporte-desormais-les-applicatio...>. The fifth result is "Hello World Node.js Example - Serverless" with a link to <https://serverless.com/framework/docs/providers/aws/.../node/>. The sixth result is "Claudia.js" with a link to <https://claudiajs.com/>. At the bottom of the page, there is a link to "A crash course on Serverless with Node.js - By Adnan Rahić".



OpenWhisk Action Annotations

Annotations enable dynamic selection of functions, like we do for OSGi services in Sling

```
/**
 * A SlingSafeMethodsServlet that renders the current Resource as simple HTML
 */
@Component(service = Servlet.class,
    name="org.apache.sling.servlets.get.DefaultGetServlet",
    property = {
        "service.description=Default GET Servlet",
        "service.vendor=The Apache Software Foundation",

        // Use this as a default servlet for Sling
        "sling.servlet.resourceTypes=sling/servlet/default",
        "sling.servlet.prefix=Integer=-1",

        // Generic handler for all get requests
        "sling.servlet.methods=GET",
        "sling.servlet.methods=HEAD"
    })
```

OSGi
service properties


```
{
  "namespace": "guest",
  "name": "somedoc-html",
  "version": "0.0.1",
  "exec": {
    "kind": "nodejs:10",
    "binary": false
  },
  "annotations": [
    {
      "key": "sling:contentType",
      "value": "text/html"
    },
    {
      "key": "sling:resourceType",
      "value": "microsling/somedoc"
    },
    {
      "key": "provide-api-key",
      "value": false
    },
    {
      "key": "sling:extensions",
      "value": "html"
    }
  ],
}
```

OpenWhisk
Annotations





Serverless μ Sling!

2007...

 / SLING-47

microsling - simple webapp to demonstrate the core principles of Sling

Details

Type:	 New Feature	Status:	CLOSED
Priority:	 Minor	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	None
Component/s:	Engine		
Labels:	None		


Description

Following our recent API redesign discussions (see <http://cwiki.apache.org/confluence/display/SLING/Sling+API+Redesign> in particular), I have started working on "microsling", a webapp that demonstrates my understanding of the "most important parts" of Sling.

Home Who? [cat /dev/brain](#) | [egrep -i 'tech|thoughts|noise'](#) [bertrand's brain grep](#)

microsling - Yet Another (cool) Web Applications Framework

October 12, 2007

From the new and improved department: I spent part of this week writing a "reduced to the max" version of the [Sling](#) core that I've called *microsling*. 

It's been a lot of fun of course, and I think the results are fairly impressive in terms of power per line of code, thanks to the power of the JCR API. Using "modern" Java, with scripting in the right places helps a lot as well, but that's *nothing new*.

In the current state, microsling allows you to use [SlingServlets](#) to process requests in a RESTful way, acting on abstract [Resources](#) and using request processing scripts in various languages.

[SlingServlets](#) are provided to create content and to render it using velocity templates or server-side javascript. (More than 1'200 lines of Java code, all inclusive.)


microsling matches the complete vision of the Sling framework, but this is already powerful stuff - and very simple to

You have searched the [bertrand's brain grep](#) blog archives for 'microsling'. If you are unable to find anything in these search results, you can try one of these links.

CAT MYSELF | SORT | HEAD -2

Bertrand Delacrétaiz here - my "résumé" will tell you more.


The opinions expressed here are my own, I'm not representing any group or company on this blog. YMMV.



FOLLOW ME ON TWITTER

microsling

LS /VAR/CATEGORIES

Select Category 

Actions selected based on Annotations

INPUT: resource type, request extension

FOR all OpenWhisk Actions **A** in this namespace,
ORDERED by preference:

GET the annotations of action **A**

IF annotations match

THEN return **A** as the Action to use

IF no Action found, use **default renderers**

sling-whiteboard: serverless-microsling/lib/openwhisk-renderer.js

```
{
  "namespace": "guest",
  "name": "somedoc-html",
  "version": "0.0.1",
  "exec": {
    "kind": "nodejs:10",
    "binary": false
  },
  "annotations": [
    {
      "key": "sling:contentType",
      "value": "text/html"
    },
    {
      "key": "sling:resourceType",
      "value": "microsling/somedoc"
    },
    {
      "key": "provide-api-key",
      "value": false
    },
    {
      "key": "sling:extensions",
      "value": "html"
    }
  ],
}
```

OpenWhisk
Annotations

DEMO: Serverless Microsling

Some Document, with a custom rendering!

This is some document

This is the somedoc-html rendering

Some Document
This is some document

```
lib
├── JS default-renderers.js
├── JS openwhisk-renderer.js
├── JS render.js
├── JS resolve-content.js
├── node_modules
├── rendering-actions
├── JS markdown-default.js
└── JS somedoc-html.js
```

```
localhost:3233/api/v1/web/guest/default/microsling/demo/index.json
JSON Raw Data Headers
Save Copy Collapse All Expand All Filter JSON
resourceType: "microsling/demo"
title: "Demo root resource"
body: "This is the content of the demo root resource"
```

Dynamic selection
Overrides
New content types





CODA

CODA

GraalVM Native Images are difficult with existing code.

Embracing the platform usually helps - worked for our prototype.

Action annotations can be used in a similar way to OSGi service properties.

Next?

- a) build on Karl & Radu's experiments for **fast-starting Sling instances** -> switch to containers -> "full" Sling?
- b) build on the **serverless microsling** prototype to create rendering pipelines -> scalable rendering?



I'm @bdelacretaz - thank you!



APACHE

OpenWhisk

