FIWARE Summit, Malaga, Spain
December 13-15, 2016

https://www.fiware.org/summit/

Open APIs for Open Minds

The ‘Serverless‘ Paradigm, OpenWhisk and FIWARE

Alex Glikson
Cloud Platforms, IBM Research
Architect, FIWARE Cloud Hosting
glikson@il.ibm.com

IBM

FIWARE

Outline

1. Overview of Serverless
2. OpenWhisk – open source ‘Serverless’ platform
3. Challenges of Serverless
4. Serverless and FIWARE

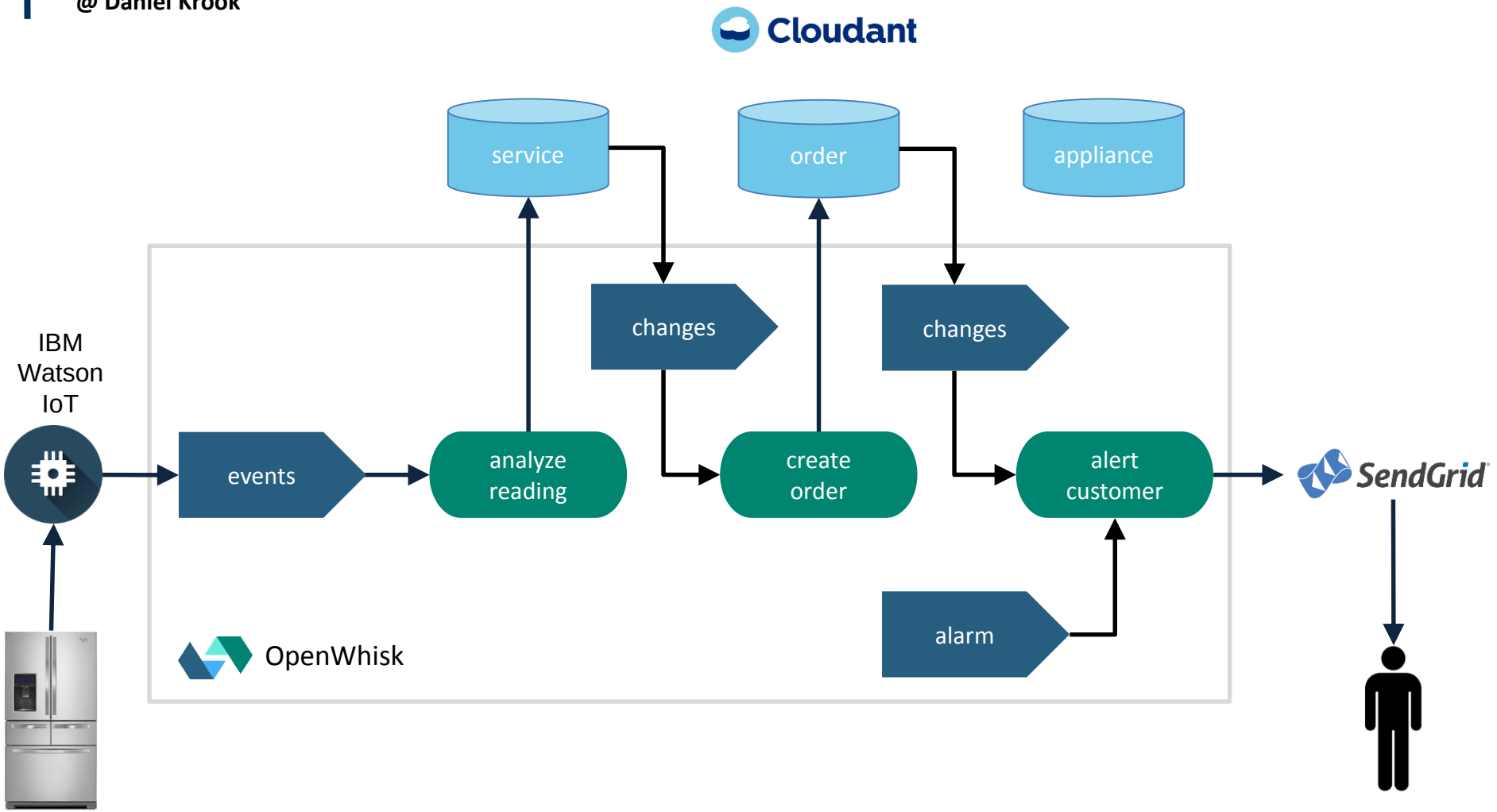
Serverless in a Nutshell

- ‘Next-gen’ PaaS (developers just write the business logic)
- for services decomposable into events/requests & handlers
- scaled, metered [and charged] by individual handler invocation

* The motivation behind the ‘serverless’ term is that the application provider doesn’t need to care about managing the underlying servers (introduced by Amazon, who didn’t have fully managed PaaS offering before Lambda)

Example: bit.ly/open-fridge

@ Daniel Krook



IBM

FIWARE

The Essence of Serverless

- What is Serverless?
  - ‘Serverless’ is a cloud-native design pattern, accompanied with a programming model and a runtime architecture
  - Aimed at radically simplified, faster and more efficient development and operation of (certain) applications
- The Pattern
  - Application is architected a set of ‘business logic’ functions, local or remote, triggered by discrete events or requests
  - The underlying runtime is (infinitely) elastic, with scaling (and chargeback) granularity of single function invocation (100ms)
  - Each local function is invoked in a sandbox, which is short-lived and ephemeral (interacting with stateful services)

Serverless Market

- Amazon Lambda
  - Pioneer of serverless, launched in Nov 2014
  - Rapid growth, dedicated mini-con at Re:Invent 2016
- Similar offerings by other commercial cloud providers
  - Google Functions, Azure Functions, IBM OpenWhisk
- Multiple niche players
  - iron.io, pubnub.com, etc
- OpenWhisk – the open source serverless platform
  - Developed by IBM, now under incubation in Apache (w/Adobe)
  - Also offered on IBM Bluemix as a fully managed service

What is serverless good for?

Serverless is good for

short-running
stateless
event-driven

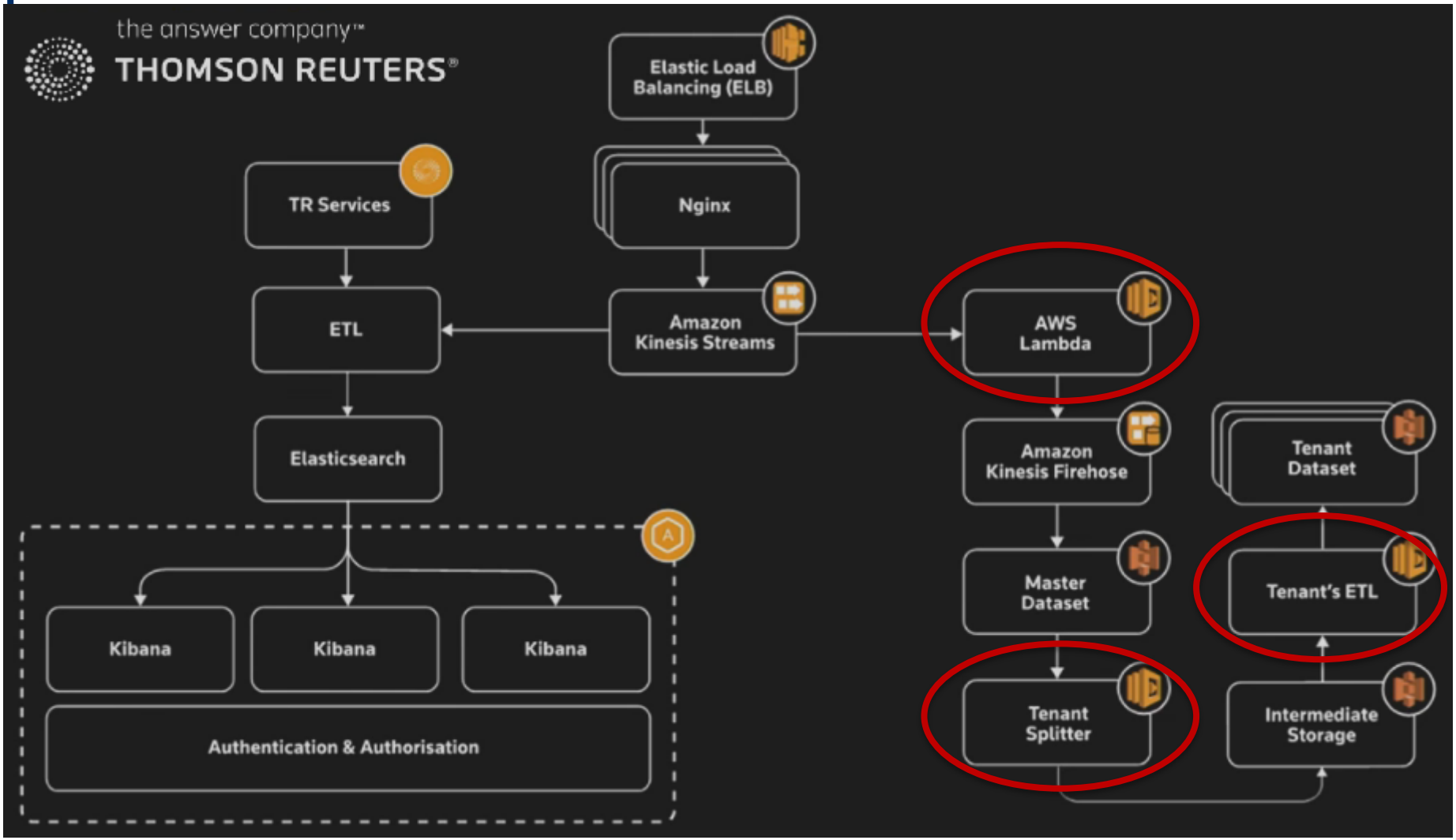
- Microservices
- Mobile Backends
- Bots, ML Inferencing
- IoT
- Data (Stream) Processing
- Cognitive
- Scatter/Gather workloads

Serverless is not good for

long-running
stateful
number crunching

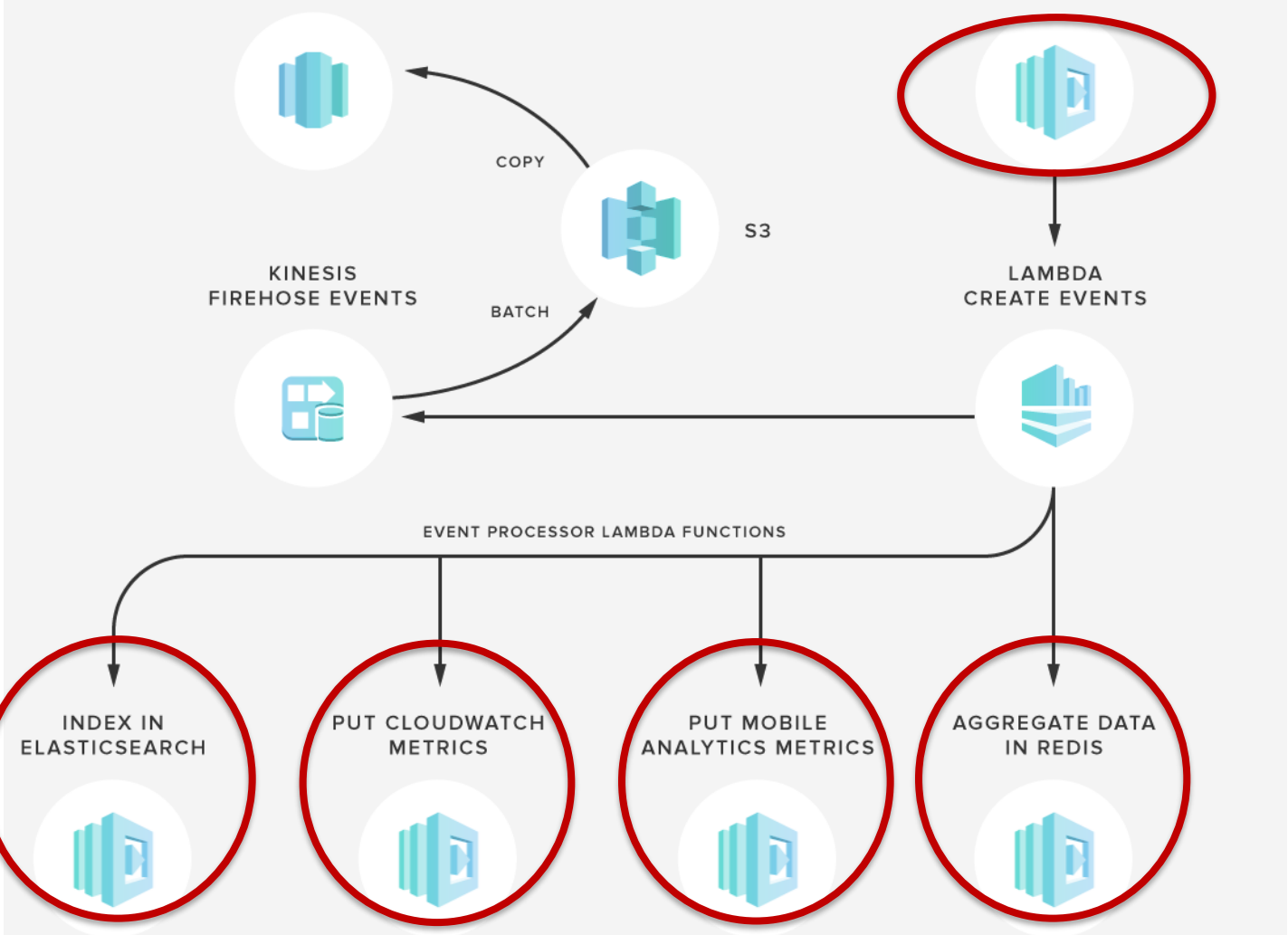
- Databases
- Deep Learning Training
- Heavy-Duty Stream Analytics
- Spark/Hadoop Analytics
- Numerical Simulation
- Video Streaming

Example: Serverless at Thomson Reuters



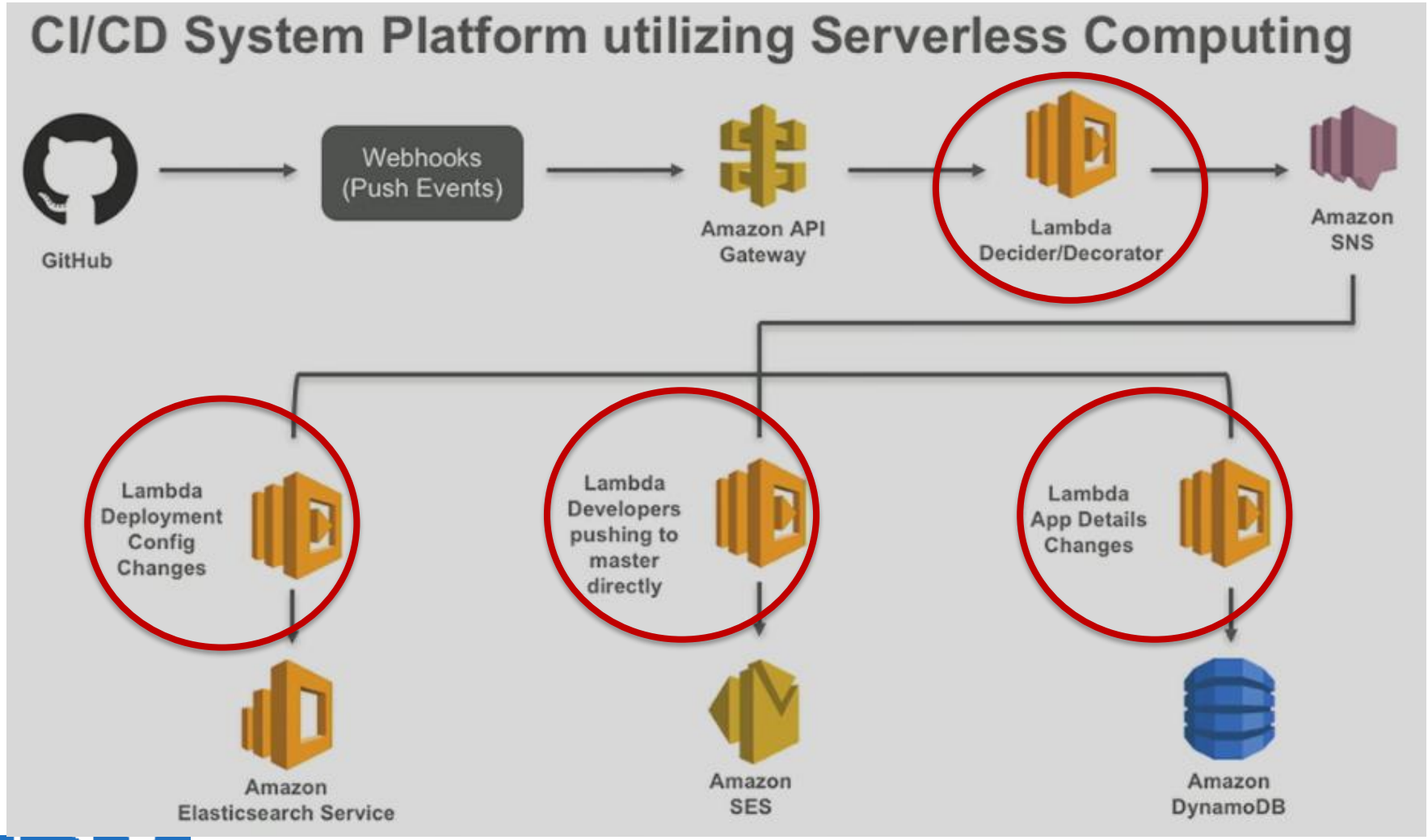
https://www.portal.reinvent.awsevents.com/connect/sessionDetail.ww?SESSION_ID=8674

Example: Serverless at Bustle



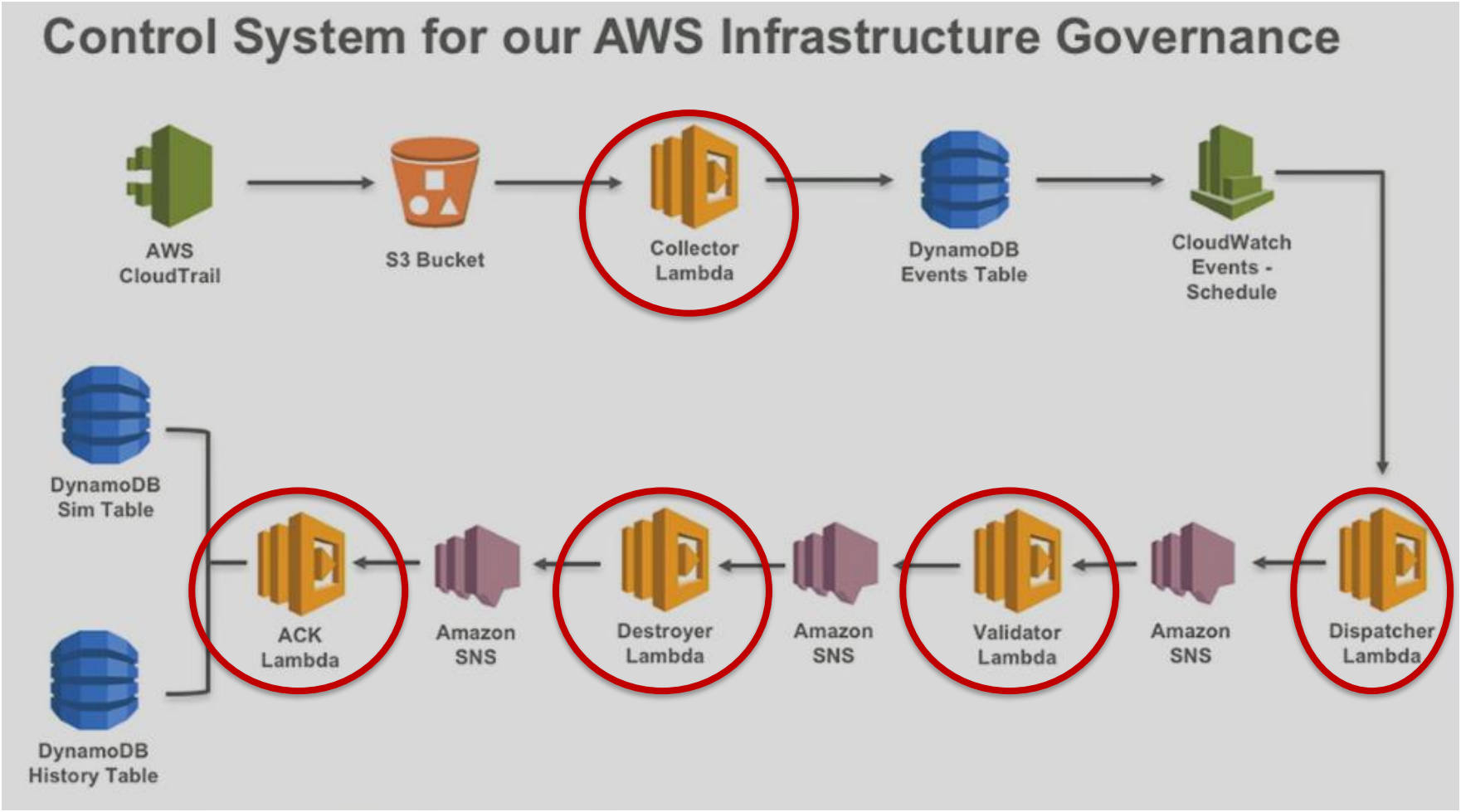
https://aws.amazon.com/solutions/case-studies/bustle/

Example: Serverless at Expedia



https://www.portal.reinvent.awsevents.com/connect/sessionDetail.ww?SESSION_ID=8671

Example: Serverless at Expedia



https://www.portal.reinvent.awsevents.com/connect/sessionDetail.ww?SESSION_ID=8671

Serverless: Why Now? The Perfect Storm

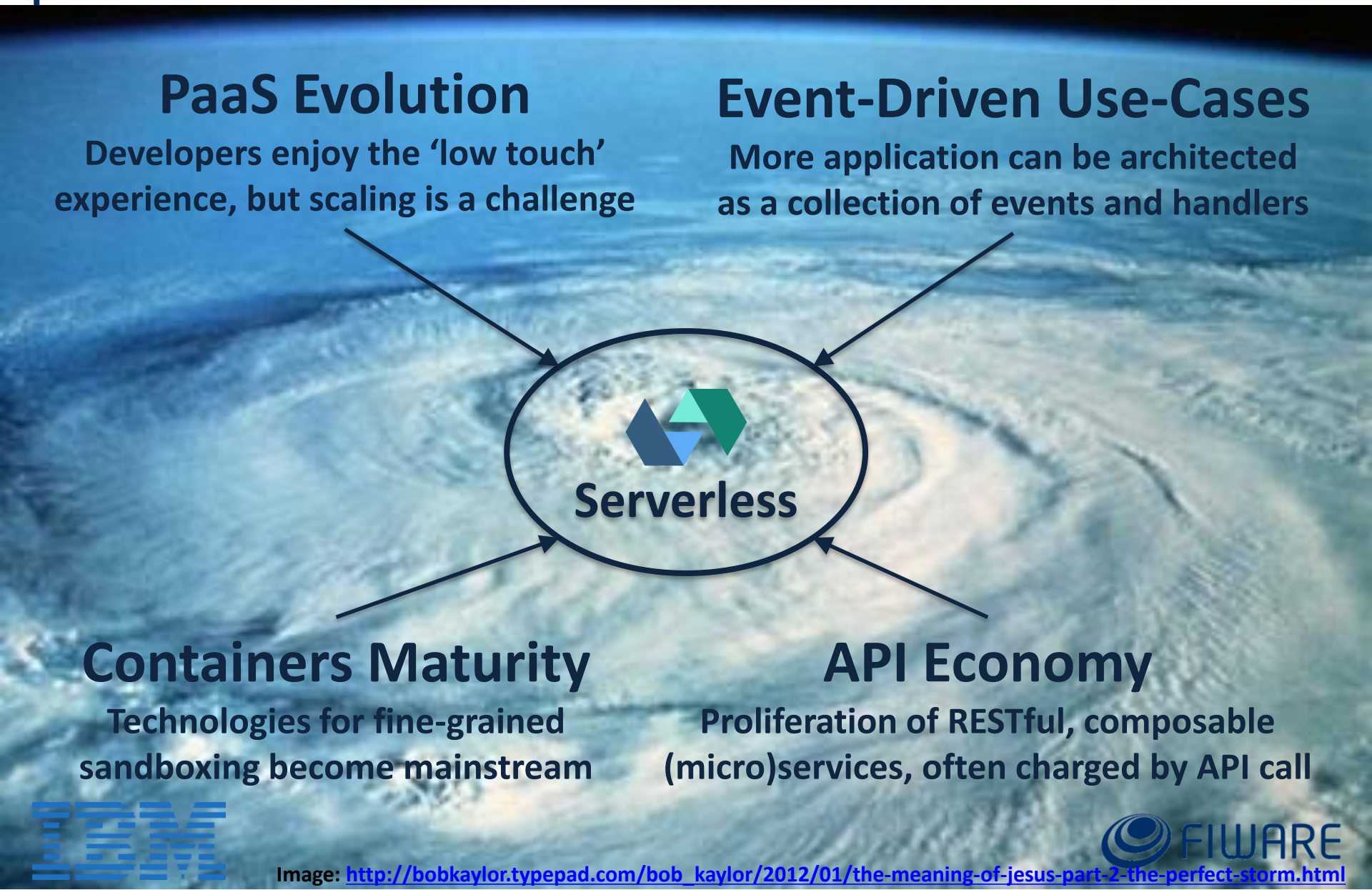
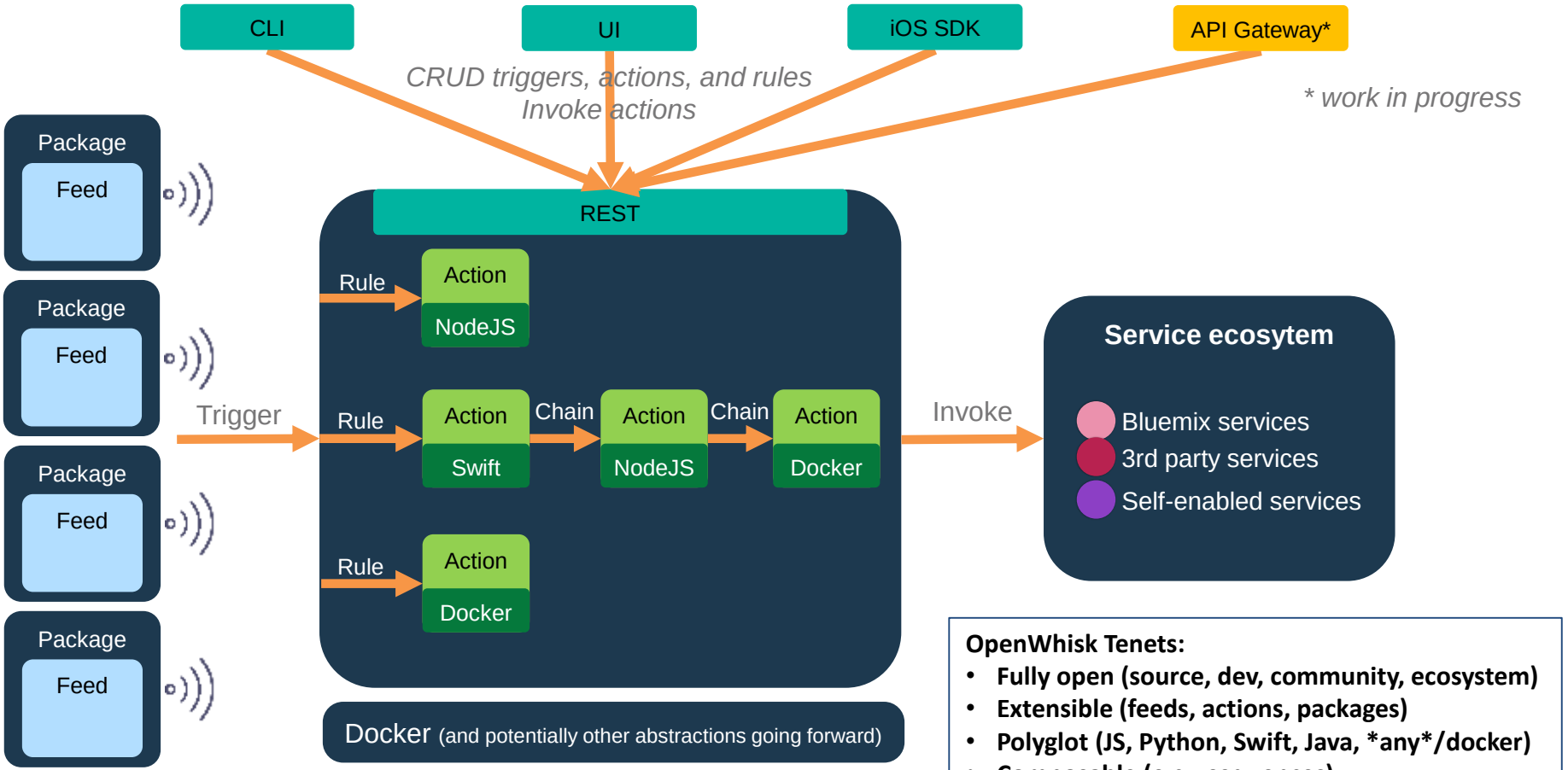


Image: http://bobkaylor.typepad.com/bob_kaylor/2012/01/the-meaning-of-jesus-part-2-the-perfect-storm.html

OpenWhisk:
Open Source Serverless Platform

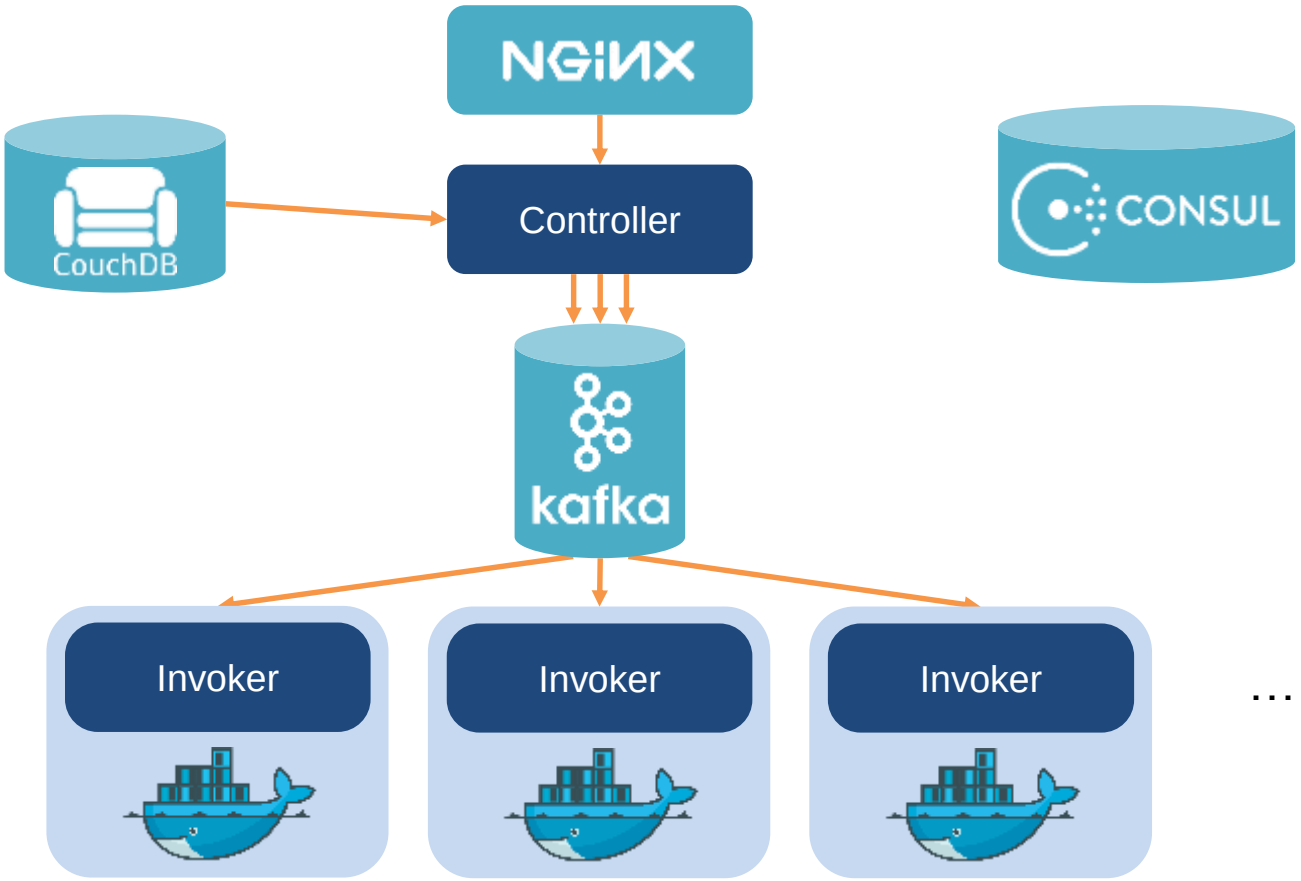


OpenWhisk Tenets:

- **Fully open (source, dev, community, ecosystem)**
- **Extensible (feeds, actions, packages)**
- **Polyglot (JS, Python, Swift, Java, *any*/docker)**
- **Composable (e.g., sequences)**
- **Per-event/request scaling, metering**

https://github.com/openwhisk

OpenWhisk Internal Architecture



OpenWhisk Catalog

- Cron
- Utils (e.g., jq)
- CouchDB/Cloudant
- Object Storage
- MQTT
- Kafka
- Node-RED
- Github
- Slack
- IBM Watson
- Weather
- WebHooks
- Mobile Push
- etc

* Some of the above are work in progress

OpenWhisk CLI: Create and test action

Create the Action that analyzes IoT readings, then stores in the database

```
wsk action create analyze-service-event analyze-service-event.js \
--param cloudant_user $CLOUDANT_USER
--param cloudant_pass $CLOUDANT_PASS
```

Invoke the Action manually with sample message data to test

```
wsk action invoke --blocking --result analyze-service-event
--param service '{"appliance_serial": "xxxxyyyyzzzz", "part_number":
"ddddeeeeffff", "reading": 13, "timestamp": 1466188262}'
```

IBM

FIWARE

OpenWhisk CLI: Link trigger to action

Create the Trigger that subscribes to an MQTT topic pattern

```
wsk trigger create openfridge-feed-trigger \
--feed mqtt/mqtt-feed-action
--param topic 'iot-2/type/+/id/+/evt/+/fmt/json'
--param url 'ssl://example.messaging.internetofthings.ibmcloud.com:8883’
```

Link the Trigger to the Action using a Rule

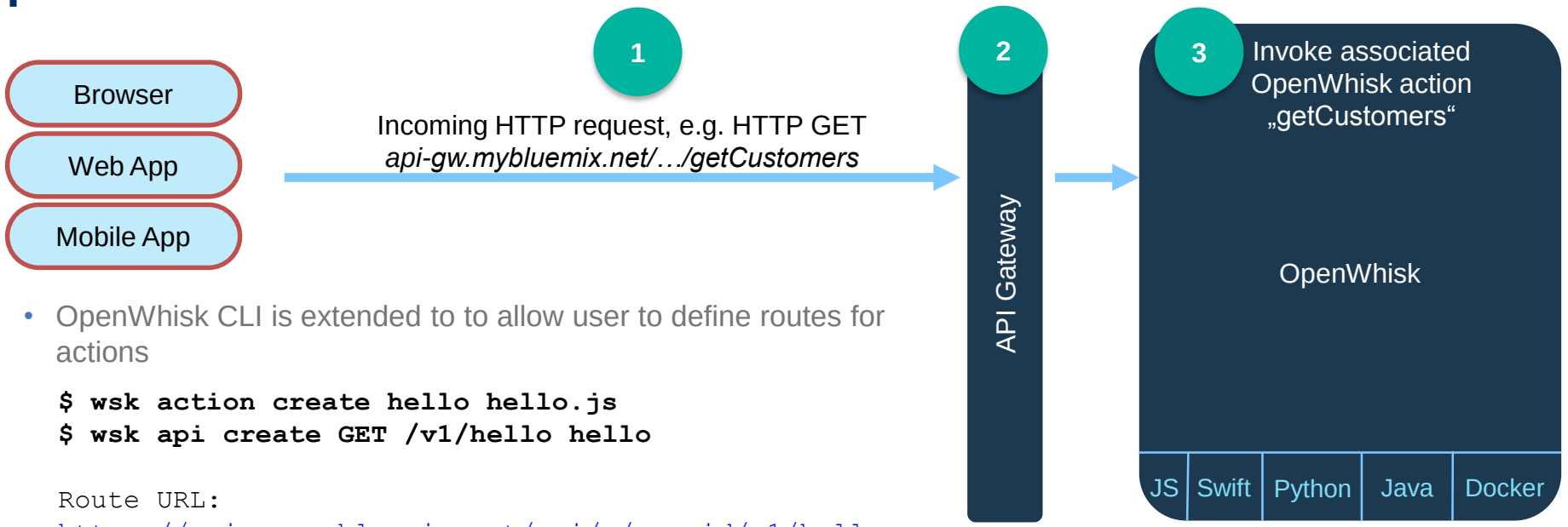
```
wsk rule create --enable openfridge-feed-rule \
openfridge-feed-trigger analyze-service-event
```

IBM

FIWARE

OpenWhisk and API Gateway

Coming soon...



- OpenWhisk CLI is extended to to allow user to define routes for actions

```
$ wsk action create hello hello.js
$ wsk api create GET /v1/hello hello

Route URL:
https://api-gw.mybluemix.net/api/ /nsuuid/v1/hello

$ curl –XGET https://api-gw.mybluemix.net/api/ /nsuuid/v1/hello
{ message: ”Hello World” }
```

- **API Gateway takes care of…**
  - **security (authenticate, authorize, threat protect, validate)**
  - **control (rate limiting, response caching)**
  - **mediation**
  - **parameter mapping**
  - **schema validation**
  - **etc**

IBM

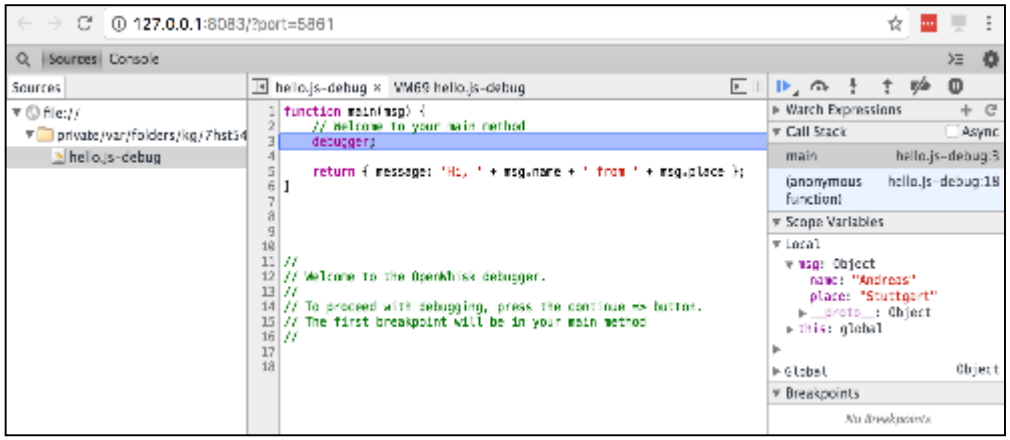
FIWARE

OpenWhisk Debugger

(https://github.com/openwhisk/openwhisk-debugger)

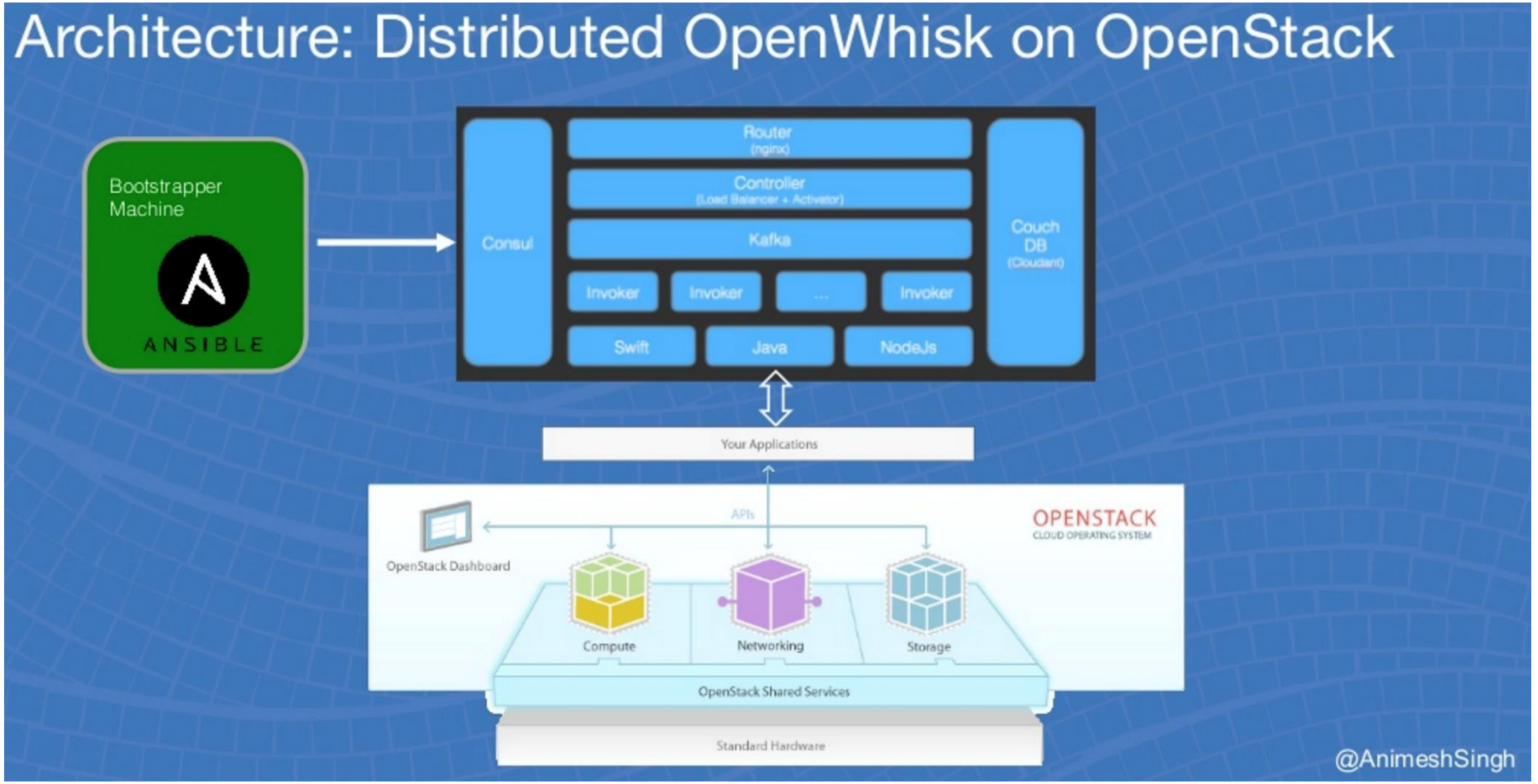
- Allows to…
  - debug actions locally
  - inspect parameter values
  - edit code & push changes
- Supports debugging…
  - NodeJS, Python and Swift actions

```
$ (wskdb) help
The available commands are:

COMMAND             DESCRIPTION
list, l             List available actions
cli                 Use the CLI debugger, when available
invoke, i           Invoke an action
inspect, ins, get   Inspect the details of an OpenWhisk action
fire, f             Fire a trigger
attach, a           Attach to an action
detach, d           Detatch from an action
diff                Show the pending diffs of a given action
publish, p          Publish pending changes to a given action
exit, quit, e, q    Quit the debugger
clean, c            Clean up debugging artifacts
create              Create an action
delete              Delete an action

help, h, ?          Print this help text
```



OpenWhisk on OpenStack

https://developer.ibm.com/openwhisk/2016/08/08/multicloud-openwhisk-build-a-distributed-openwhisk-deployment-on-openstack/



Challenges of Serverless

- Opinionated programming model
  - Aligned with 12-factor approach to cloud-native applications
- Per-invocation latency & overhead
- High-performance persistent state
- Life cycle of composite serverless applications
- Monitoring, error handling, testing, debugging

Proposal: Serverless/OpenWhisk in FIWARE

- FIWARE is an open source platform, community and ecosystem for context-aware cloud-based applications enabled by Big Data and IoT
  - OpenWhisk shares similar open principles and philosophy
- Serverless capabilities (based on OpenWhisk) can simplify development and hosting of FIWARE applications, e.g.,
  - Simple interface to develop and run actions (in various programming languages), integrated into FIWARE user experience
    - FIWARE WireCloud integration/frontend?
  - OpenWhisk packages surfacing APIs of individual FIWARE Generic Enablers (GEs) as OpenWhisk triggers and actions
  - FIWARE Orion/NGSI package encapsulating generic context operations (queries, updates), subscription feed
    - Feed provider via Cygnus ➔ OpenWhisk bridge
  - GE developers could take advantage of OpenWhisk too

https://www.fiware.org/

Thank you!

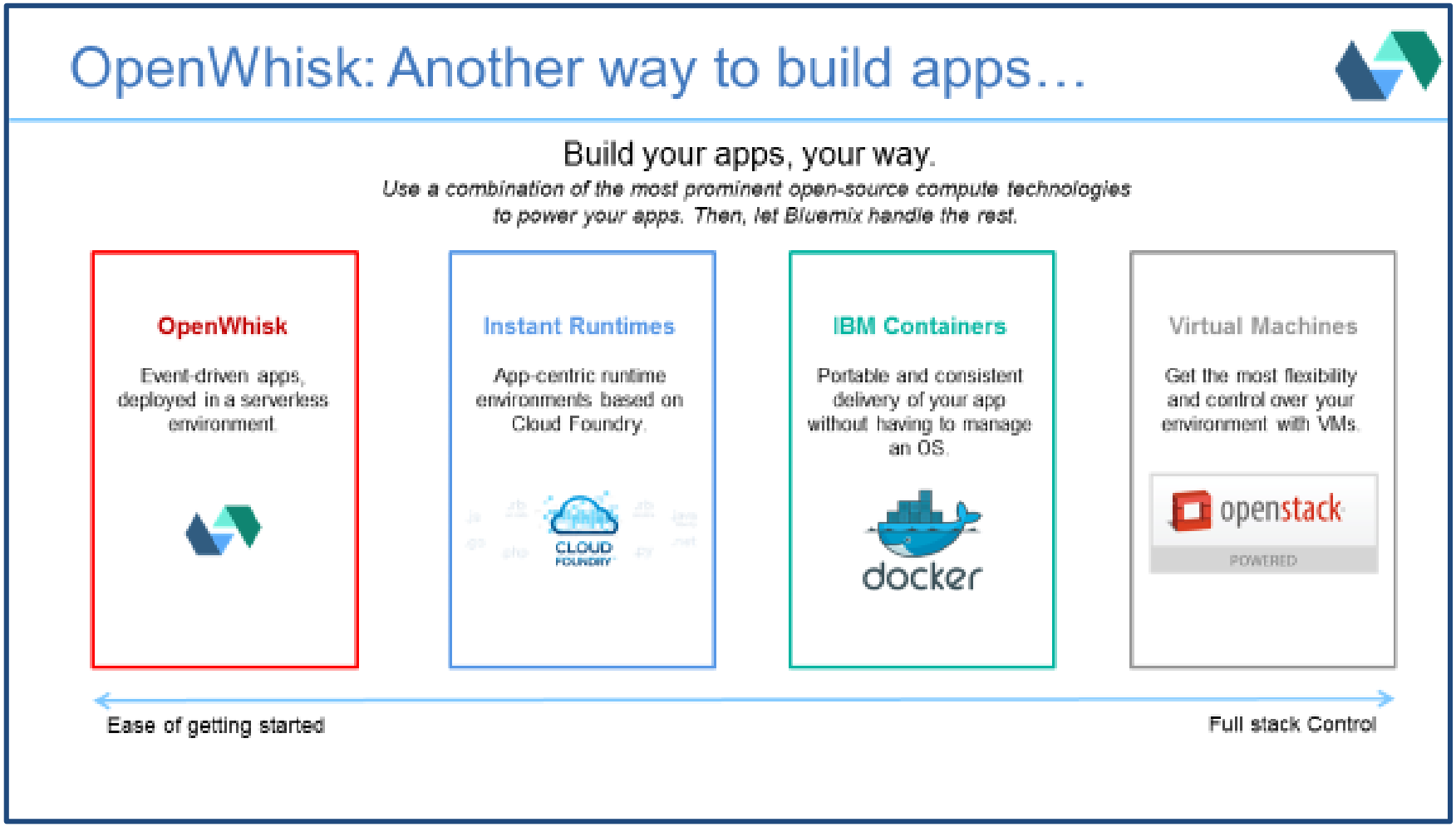
Alex Glikson
Cloud Platforms, IBM Research
Architect, FIWARE Cloud Hosting
glikson@il.ibm.com

http://fiware.org
Follow @FIWARE on Twitter

IBM

FIWARE

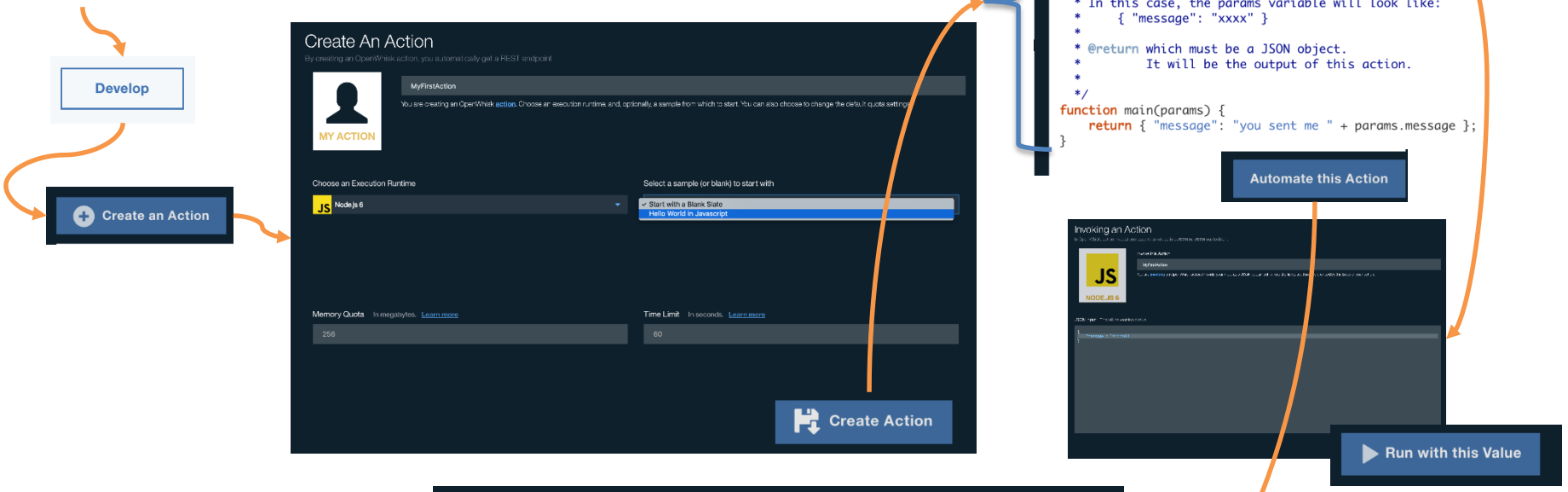
OpenWhisk on IBM Bluemix



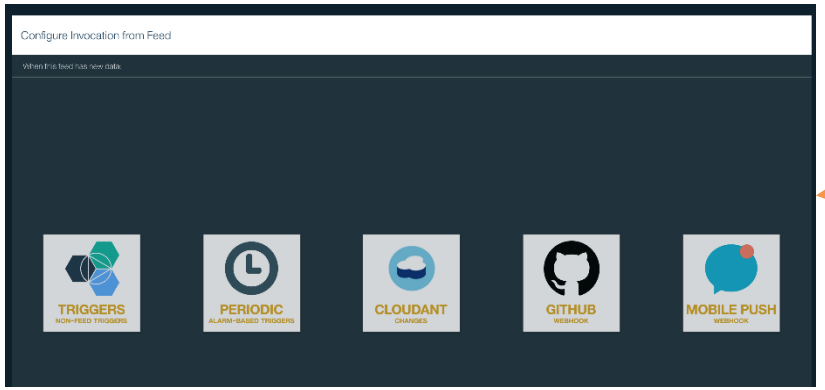
Getting started with OpenWhisk in Bluemix

Click here and run your first action in 30 secs:

https://console.ng.bluemix.net/openwhisk/

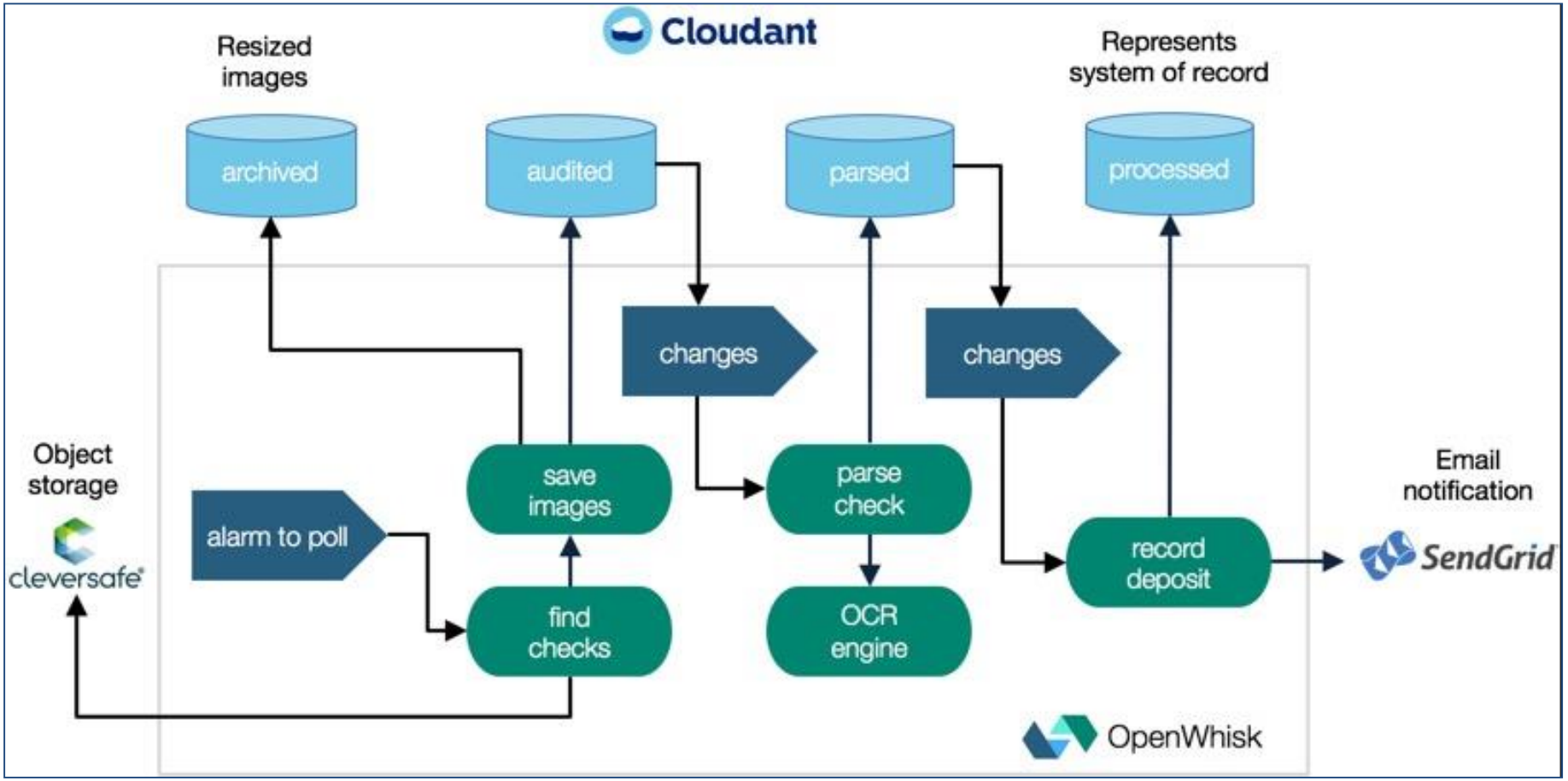


Associate an action with event triggers:



Serverless check processing with OpenWhisk

https://github.com/krook/openchecks



IBM

FIWARE

Serverless can handle many cloud native app 12 Factors

I	**Codebase**	**Handled by developer** (Manage versioning of functions on their own)
II	**Dependencies**	**Handled by developer**, facilitated by serverless platform (Runtimes and packages)
III	**Config**	**Handled by platform** (Environment variables or injected event parameters)
IV	**Backing services**	**Handled by platform** (Connection information injected as event parameters)
V	**Build, release, run**	**Handled by platform** (Deployed resources are immutable and internally versioned)
VI	**Processes**	**Handled by platform** (Single stateless containers often used)
VII	**Port binding**	**Handled by platform** (Actions or functions are automatically discovered)
VIII	**Concurrency**	**Handled by platform** (Process model is hidden and scales in response to demand)
IX	**Disposability**	**Handled by platform** (Lifecycle is hidden from the user, fast startup and elastic scale is prioritized)
X	**Dev/prod parity**	**Handled by developer** (The developer is the deployer. Scope of what differs is narrower)
XI	**Logs**	**Handled by platform** (Developer writes to console.log, platform handles log streaming)
XII	**Admin processes**	**Handled by developer** (No distinction between one off processes and long running)