Paremus

Event Driven IoT - Serverless Functions for OSGi

Tim Ward
tim.ward@paremus.com

Who is Tim Ward?

Chief Technology Officer at Paremus

10+ years developing OSGi specifications

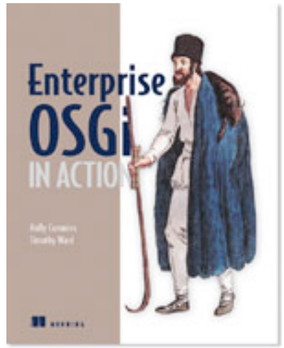
Co-chair of the OSGi IoT Expert Group

Interested in Asynchronous Distributed Systems

Author of Manning's Enterprise OSGi in Action

http://www.manning.com/cummins

@TimothyWard



Serverless Computing

Serverless Computing - A Primer

- Serverless Computing is a “Cloud Service”
  - You write and deploy the “application” code
  - The Cloud provider hosts, scales, and maintains the infrastructure
- You can think of Serverless as a “half-way house”
  - Less maintenance overhead (for you) than “Platform as a Service”
  - You can deploy your own code, unlike “Software as a Service”
- Different Providers have different API requirements

Function as a Service

- Serverless applications are usually stateless
  - This is typically known as “Function at a Service”
  - You can use a Data Store (usually provided by your Cloud Provider)
- Some providers let you store “temporary state”
  - A small amount of disk and/or memory shared between executions
  - This can be thrown away by the provider for any reason!
- Define an entry point and “triggers”
  - Http Request, Changes to S3 bucket, MQTT message…

Why is this a good thing for IoT?

- Functions are easy to deploy
  - Cloud Providers have deployment APIs - no scripts to write/maintain
  - Platform updates and security fixes are automatic
- Flexibility of Input and scaling
  - Multiple IoT Device inputs can map to the same trigger
  - Bursts of Events can be handled by scaling the function
- Costs less money (you hope!)
  - “Pay per invocation” when function gets called
  - Using more CPU/Memory increases the cost per call
  - Unexpected load means unexpected bills!

Java Examples

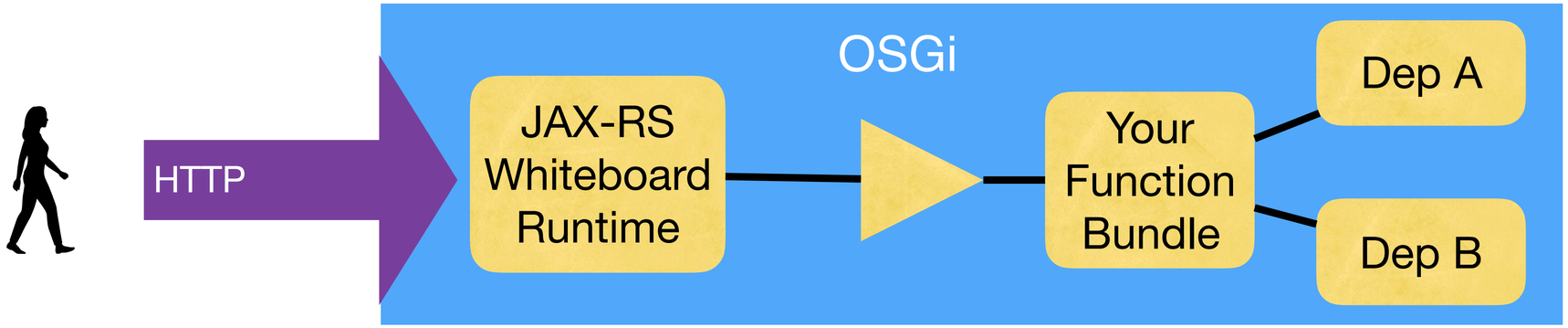
AWS Lambda (Java)

- The “Application” is a JAR file
- “Entry Point” is a method
  - Can implement an interface or be reflectively called
- Dependencies must be packaged into the deployment
  - Recommended to use “shading” to avoid clashes with the runtime
- “Entry Point” and “Trigger” defined in a descriptor file
- Different Providers have different API requirements
  - But broadly similar in Azure, Google, Apache OpenWhisk…

Does this sound familiar?

- Serverless Functions have
  - Nominated entry point
  - Metadata packaged to describe the function
  - Internal Dependencies “managed” separately from the platform

- OSGi services can be good candidates for Functions!
  - OSGi’s Whiteboard pattern is an example of a serverless architecture



OSGi “Function as a Service”

- The “Application” is a Bundle (JAR file)
  - “Entry Point” is a service
  - Dependencies Defined in a manifest

- So for HTTP input…

```
@Component(service=Upper.class)
@JaxrsResource
public class Upper {

    @Path("rest/upper/{param}")
    @GET
    public String toUpper(@PathParam("param") String param) {

        return param.toUpperCase();
    }
}
```

Are we done already?

- The example on the previous slide is great for HTTP
  - But what if our input came in from some other source?
- In IoT data comes from all over the place
  - And using *many* different protocols
- OSGi has made it really easy to plug in connectors
  - But should you need to deal with different protocols
  - How many things do you have to write?

Being protocol independent

- Good Functions should be protocol independent
  - The function shouldn’t know or care about HTTP/MQTT…
- Good functions should accept typed data
  - The input parameters should define a schema
- Good functions should not directly call other functions
  - Compose functions through trigger events
  - Better scaling if the function is already running somewhere

Using Functions in OSGi

BRAIN-IoT - Intelligent Decentralised Fog Platform

BRAIN-IoT

model-Based fRamework for dependable sensing and Actuation in INtelligent decentralized IoT systems

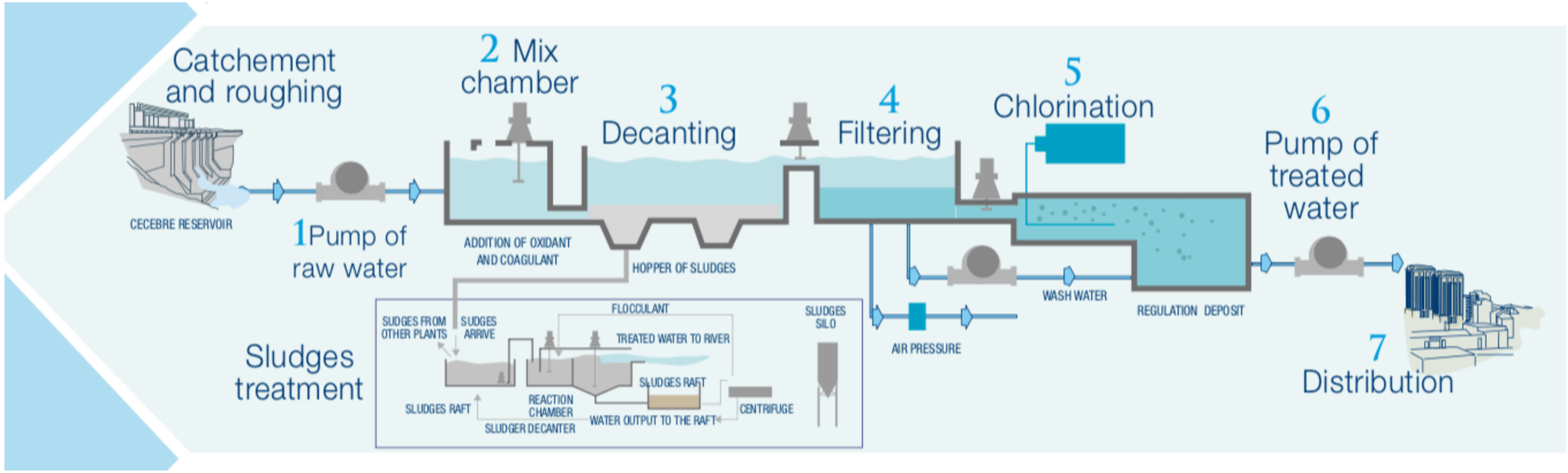
European Commission

HORIZON 2020

Part of the European Union Horizon 2020 program

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780089.

BRAIN-IoT: Mission Critical City Infrastructure



- Mission Critical Water Utility Infrastructure: location A Coruña, Spain
- Real Time Sensing, AI anomaly detection and Actuation across a sophisticated physical City Infrastructure.
- Robustness and Security paramount.

Architecture of the BRAIN-IoT project

- BRAIN-IoT is built from interacting “Smart Behaviours”
  - A Smart Behaviour consumes and sends “data” events
  - Events may be generated by sensors, users, or by other Smart Behaviours
- A Smart Behaviour can be implemented any way you like
  - Each Behaviour is its own independent decoupled unit
- Smart Behaviours use typed event data
  - This makes it much simpler to develop
- “Unhandled” Events are processed by a “special” behaviour
  - This is used to trigger the dynamic deployment of new Smart Behaviours

OSGi Event Admin

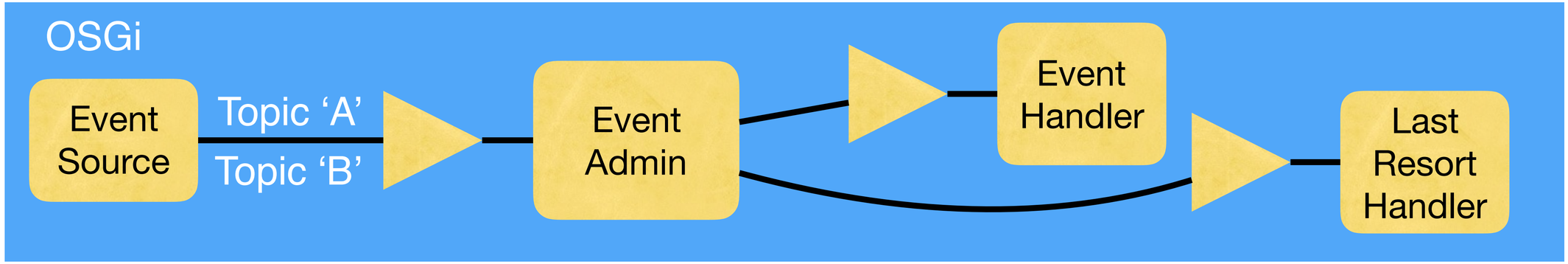
- Event Admin delivers data to Event Handler services
  - Whiteboard Functions with a defined entry point ✔
- It's easy to map inputs from different sources to events
  - Protocol Independence ✔
- Event Handlers (functions) can also send events
  - Decoupled triggering of other functions ✔
- The Event Admin specification is very mature
  - But also old - Events are opaque Maps of String to Object ✘

Updating OSGi's Event Admin

- BRAIN-IoT needs the good things from Event Admin
  - Loosely Coupled Whiteboard ✔
  - Asynchronous Delivery ✔
- But also the things that are missing
  - Strongly Typed Event data ✘
  - Monitoring of Event Delivery ✘
  - Notification of "unhandled" events ✘

Updating OSGi's Event Admin (2)

- Update the Event Admin API to be Type Safe
  - Use OSGi Data Transfer Objects
- Keep the concept of event "topics"
  - Allow the topic to be defaulted based on the type of the event DTO
- Add support for "Handlers of Last Resort"
  - An "untyped" handler called when there is no real handler for an event



Updating OSGi's Event Admin (3)

- Allow schema conversion
  - The Consumer can have a custom view of the sender's data
- Add a new Event Monitor API
  - Using a "promiscuous" Event Handler disturbs the system
  - Make use of OSGi PushStreams to provide live event flow data
- Push this back into the OSGi Alliance!
  - https://github.com/osgi/design/tree/master/rfcs/rfc0244
- The proposed changes are compatibility breaking
  - Likely to be a new specification rather than a version 2.0

Putting it all together (Demo!)

Paremus

Tim Ward

tim.ward@paremus.com

www.paremus.com @Paremus info@paremus.com

eclipsecon Europe 2019

OSGi™ Community Event 2019

LUDWIGSBURG, GERMANY | OCTOBER 21 - 24, 2019

EVALUATE THE SESSIONS

Sign in and vote using the conference app or eclipsecon.org

-1 0 +1