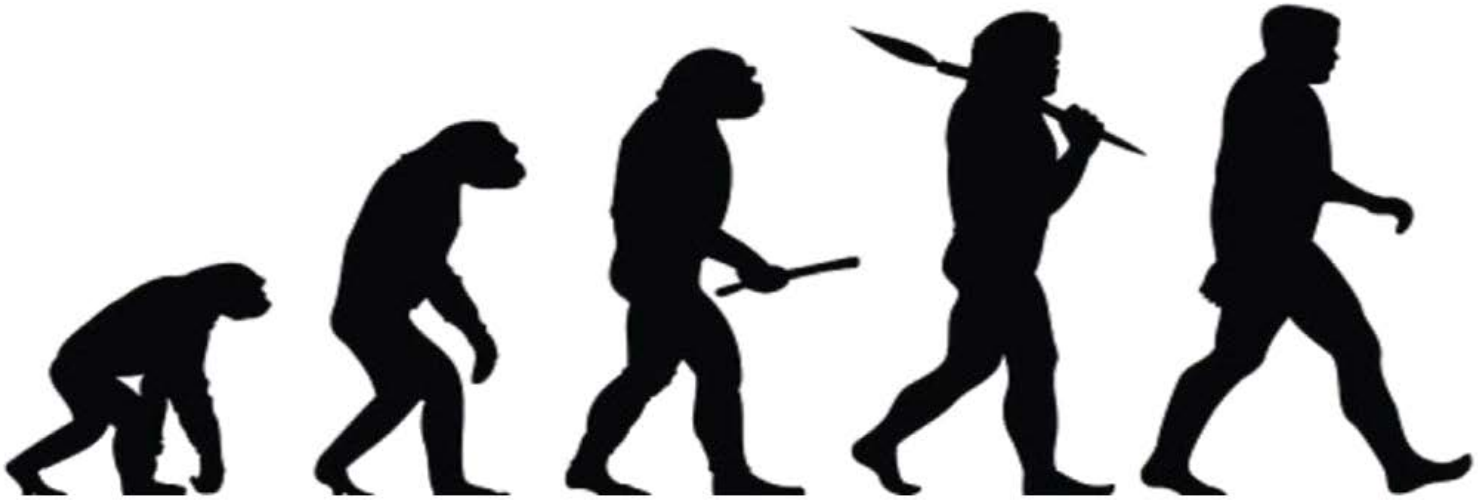
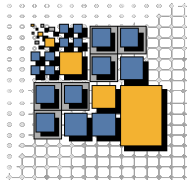


Source: https://hackernoon.com/how-can-serverless-computing-benefit-your-startup-67503e08f76e, last accessed June 2018



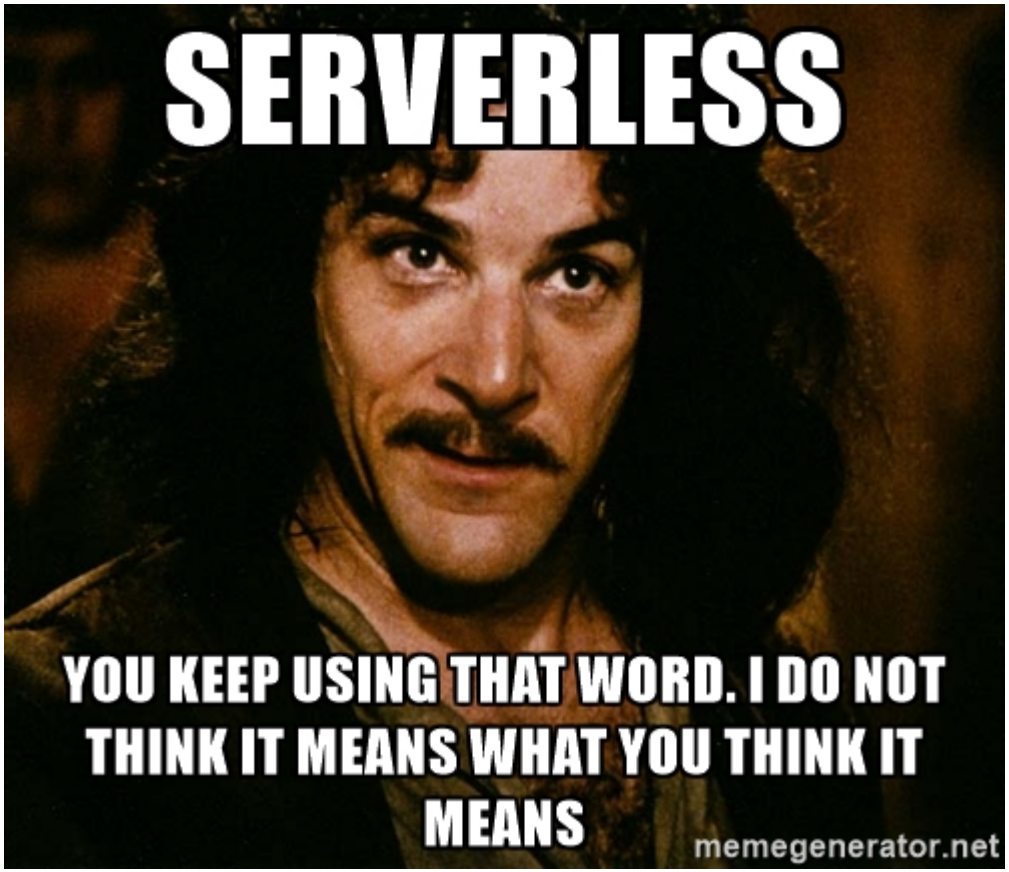
Troubleshooting Serverless Functions

A Combined Monitoring and Debugging Approach

Johannes Manner · Stefan Kolb · Guido Wirtz
Distributed Systems Group
Otto-Friedrich-University Bamberg, Germany

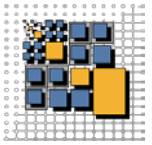
Serverless

- Majority of authors implicitly address Function as-a-Service (FaaS), when talking about Serverless



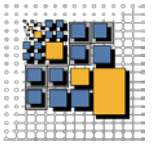
Source: http://tekhead.it/blog/2016/06/does-a-serverless-brexit-mean-goodbye-to-infrastructure-management-problems/

FaaS Characteristics



- Stateless, event-driven, ephemeral functions
- Abstraction of (all) operational tasks
- Auto scaling on demand
- Scaling to zero
- Pay per use billing model

Situation – Problem - Contribution



Situation

- “For deployed functions, you depend heavily on the logs you create to inform an investigation of function behavior.”
  (AWS: Serverless Architectures with AWS Lambda: Overview and Best Practices, https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf, page 31)
- Unstructured log messages (only plain text and metadata)
- Unit testing to guarantee functional correctness

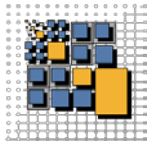
Problem

- Developer searches chaotically in log files to extract information
- Parameters are often missing (insufficient information quality)
- No mature tooling to support cloud function debugging

Contribution

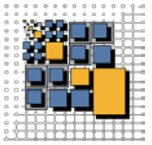
- Logging input, context and output enables a posteriori debugging
- Define agreed upon, machine readable format for log messages
- Automate test generation via template approach

Cloud Function Troubleshooting Lifecycle (1/6)



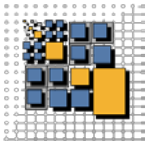
Offline (Developer's machine) Online (FaaS platform)

Implementation – Calculator



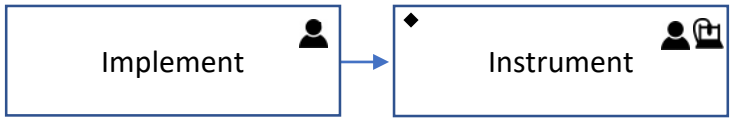
```
class Calculator {
    public int calculate(CalculatorInput input) {
        return input.getX() / input.getY();
    }
}

class CalculatorInput {
    int x;
    int y;
    int getX() {
        return x;
    }
    int getY() {
        return x;
    }
    void setX(int x) {
        this.x = x;
    }
    void setY(int y) {
        this.y = y;
    }
}
```

Implementation – AWS Lambda Handler



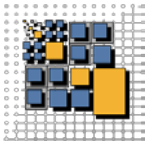
```
public class CalculatorHandler
    implements RequestHandler<CalculatorInput, Integer> {

    @Override
    public Integer handleRequest(CalculatorInput input, Context context) {

        Calculator calc = new Calculator();
        Integer result = calc.calculate(input);

        return result;
    }
}
```

Cloud Function Troubleshooting Lifecycle (2/6)



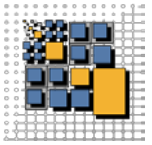
Offline (Developer's machine) Online (FaaS platform)

Implementation – AWS Lambda Handler



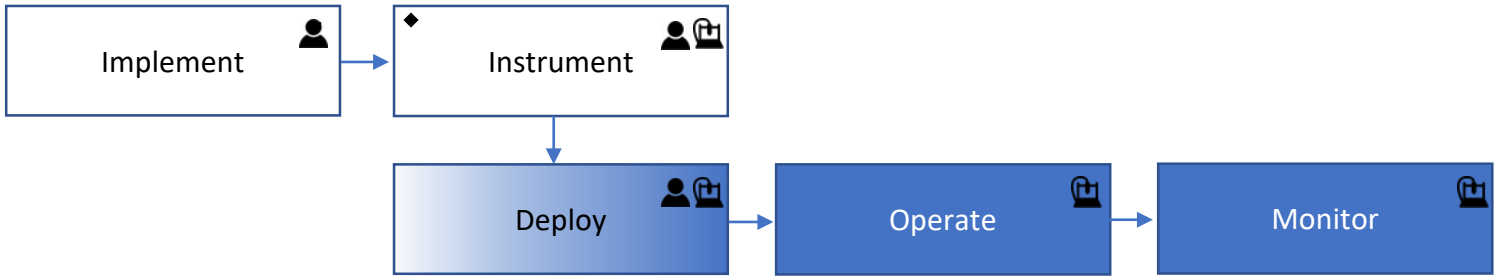
```
public class CalculatorHandler
    implements RequestHandler<CalculatorInput, Integer> {

    @Override
    public Integer handleRequest(CalculatorInput input, Context context) {

        Calculator calc = new Calculator();
        Integer result = calc.calculate(input);

        return result;
    }
}
```

Implementation – AWS Lambda Handler



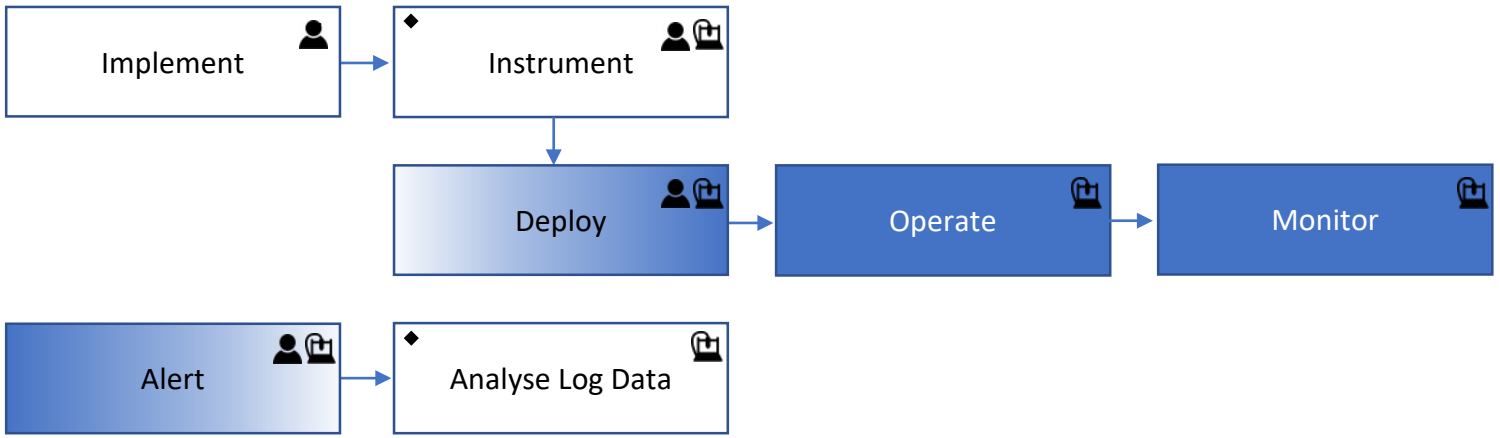
```
public class CalculatorHandler
    implements RequestHandler<CalculatorInput, Integer> {

    @Override
    public Integer handleRequest(CalculatorInput input, Context context) {

        SeMoDeInstrumentation.instrumentFunction("Calculator", "calculate",
                "CalculatorInput", input, "Integer", context.getLogger());

        Calculator calc = new Calculator();
        Integer result = calc.calculate(input);

        SeMoDeInstrumentation.instrumentFunction(result, context.getLogger());

        return result;
    }
}
```

Cloud Function Troubleshooting Lifecycle (3/6)



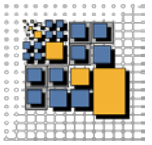
Offline (Developer's machine) Online (FaaS platform)

Cloud Function Troubleshooting Lifecycle (4/6)



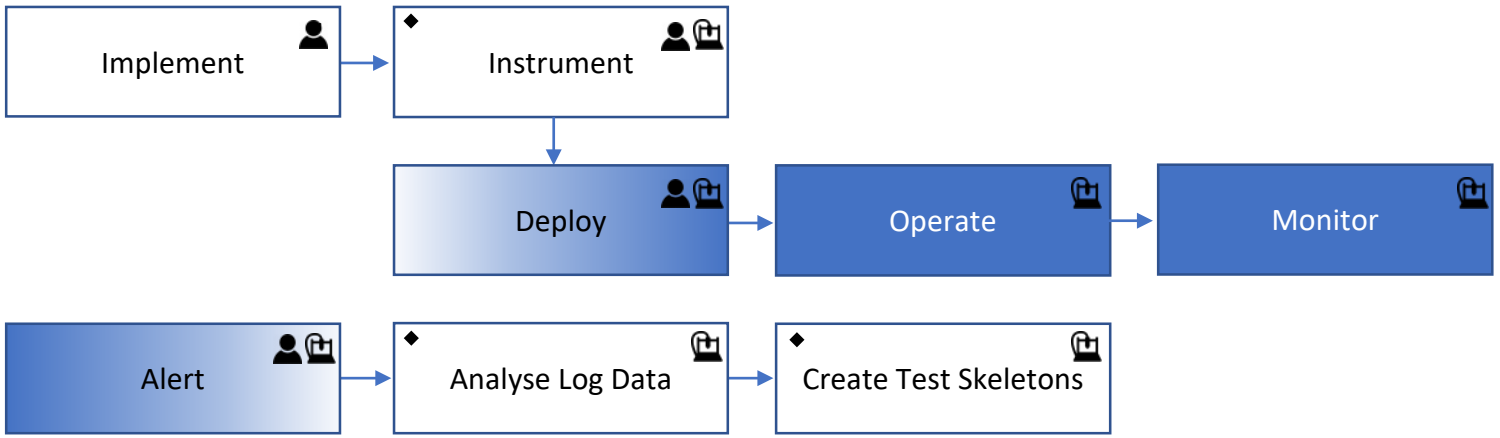
Offline (Developer's machine) Online (FaaS platform)

Cloud Function Troubleshooting Lifecycle



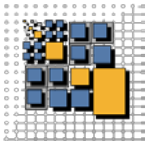
```
13:11:23 START RequestId: c336d61e-761d-11e8-85ef-8f91ece55e Version: $LATEST
13:11:23 TroubleshootLambda::handleRequest::HANDLERCLASS::Calculator
13:11:23 TroubleshootLambda::handleRequest::HANDLERMETHOD::calculate
13:11:23 TroubleshootLambda::handleRequest::INPUTCLASS::CalculatorInput
13:11:23 TroubleshootLambda::handleRequest::INPUTJSON::{"x":4,"y":0}
13:11:23 TroubleshootLambda::handleRequest::OUTPUTCLASS::Integer
13:11:23 / by zero: java.lang.ArithmeticException java.lang.ArithmeticException:
         summer.soc.demo.calculator.Calculator.calculate(CalculatorHandler.java:26)
         at summer.soc.demo.calculator.CalculatorHandler.handleRequest
         (CalculatorHandler.java:16) at summer.soc.demo.calculator.CalculatorHandler
         .handleRequest(CalculatorHandler.java:1)
13:11:24 END RequestId: c336d61e-761d-11e8-85ef-8f91ece55e
13:11:24 REPORT RequestId: c336d61e-761d-11e8-85ef-8f91ece55e Duration: 438.38 ms
         Billed Duration: 500 ms Memory Size: 512 MB Max Memory Used: 75 MB
```

Cloud Function Troubleshooting Lifecycle (5/6)



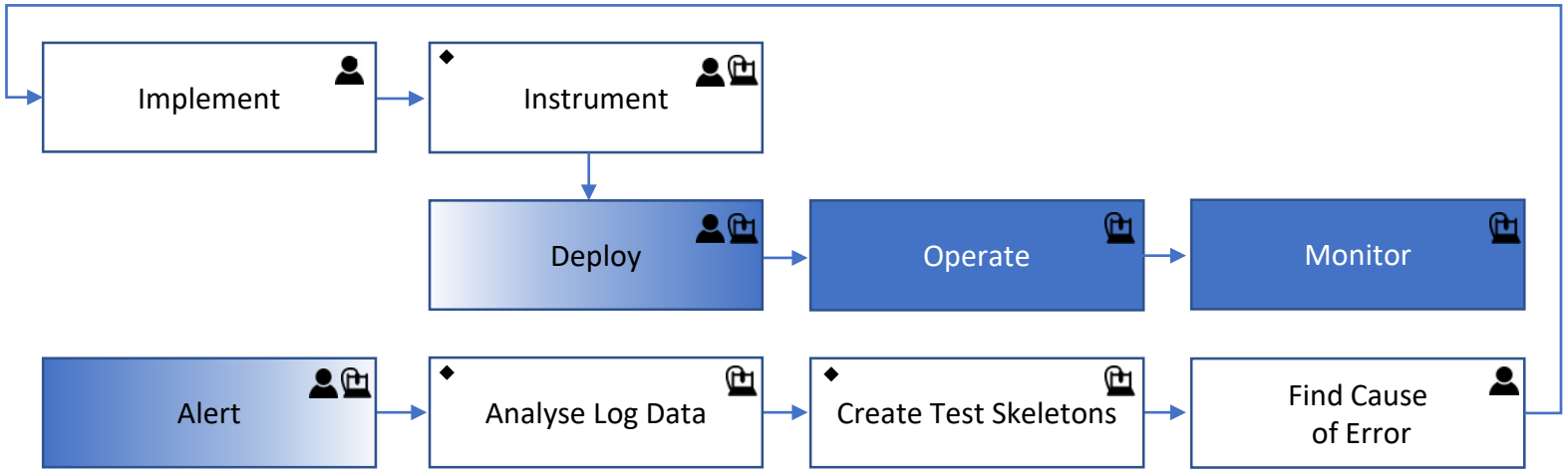
Offline (Developer's machine) Online (FaaS platform)

Cloud Function Troubleshooting Lifecycle



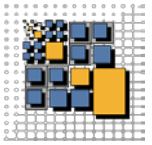
```
public class Calculator_c336d61e_761d_11e8_85ef_8f91ece55e {

    private static CalculatorInput input;

    @BeforeClass
    public static void createInput() throws IOException {
        String jsonInput = "{\""
                             + "x\":4,"
                           + "\"y\":0"
                         + "}";
        input = new ObjectMapper().readValue(jsonInput, CalculatorInput.class);
    }

    @Test
    public void testLambdaFunctionHandler() {
        Calculator handler = new Calculator();

        Integer output = handler.calculate(input);

        // TODO: validate output here
        Assert.assertEquals(??, output);
    }
}
```

Cloud Function Troubleshooting Lifecycle (6/6)



Offline (Developer's machine) Online (FaaS platform)

Assessment: Advantages - Drawbacks



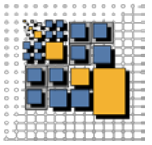
Advantages

- Approach decouples debugging process and FaaS platform
- Developer can work with mature, accustomed tools
- Higher test coverage
- Increase in log information quality

Drawbacks

- Test case duplication (single error in your code, results in N faulty executions and N test cases generated by using SeMoDe)
- Logging extend execution time of cloud functions (Cost ⇑)

Future Work – Related to Presented Work



- Introduced performance and cost drawbacks
- Test de-duplication (concept)
- Enhance SeMoDe‘s platform, language and test de-duplication support

Dissertation Plan – Next Steps

- Quantitative cold start investigation - comparison to VM, CaaS, PaaS
  - Based on user scenarios with an orchestration of small service
  - Literature work on VM, CaaS and PaaS w.r.t. engineering decisions

- Decision criteria to determine, which memory setting is the best trade-off between cost and performance of cloud function

I'm very interested to discuss the presented work and the planned research objects to get your feedback during the poster session ☺

Contact

Johannes Manner

/johannes-manner

johannes.manner@uni-bamberg.de