Imperial College London

@rchatley #vss19

Nimbus: Improving Developer Productivity for Function-as-a-Service

Robert Chatley **Thomas Allerton**

Work partially supported by the RADON project

RADON

http://radon-h2020.eu

The Developer Experience

CloudFormation

serverless

Thomas worked on developing a backend system using AWS Lambda. He found it painful.

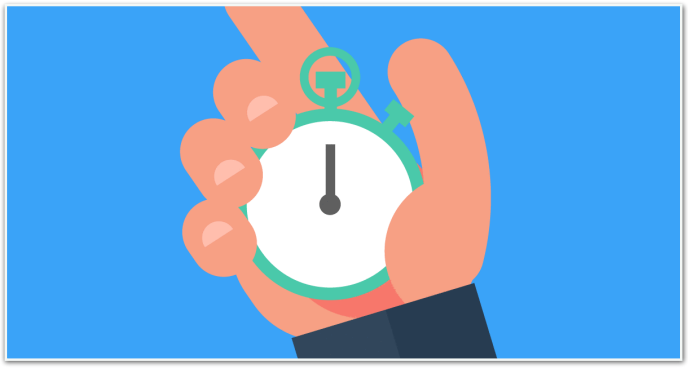
The Developer Experience

```
private APIGatewayProxyResponseEvent register(APIGatewayProxyRequestEvent event,
Context context) {

   String username = event.getBody();

   AmazonDynamoDB client = AmazonDynamoDBClientBuilder.defaultClient();

   DynamoDB dynamoDb = new DynamoDB(client);

   Table table = dynamoDb.getTable("UserDetail");

   table.putItem(new Item().withString("username", username));

   return new APIGatewayProxyResponseEvent().withStatusCode(200).withBody("");

}
```

A lot of boiler plate and cloud-specific code to write.

Also a lot of configuration (e.g. CloudFormation).

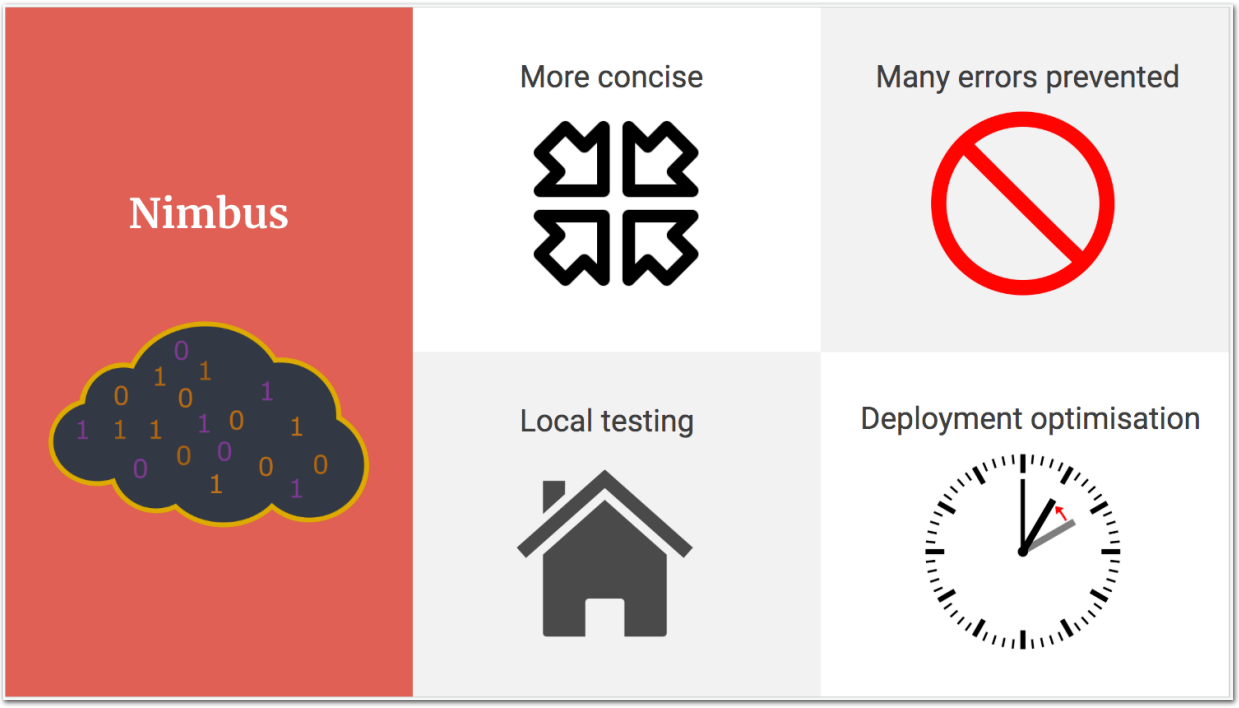
The Developer Experience



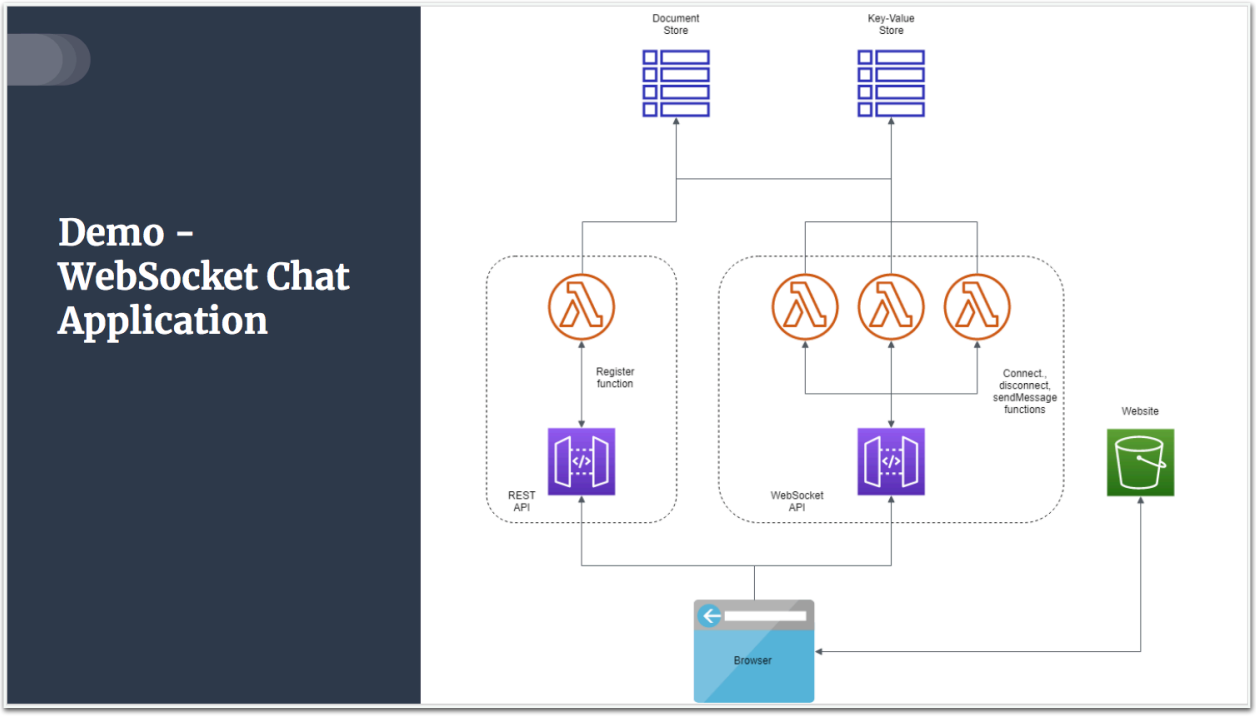
Config errors found only during deployment

Deployment can take minutes

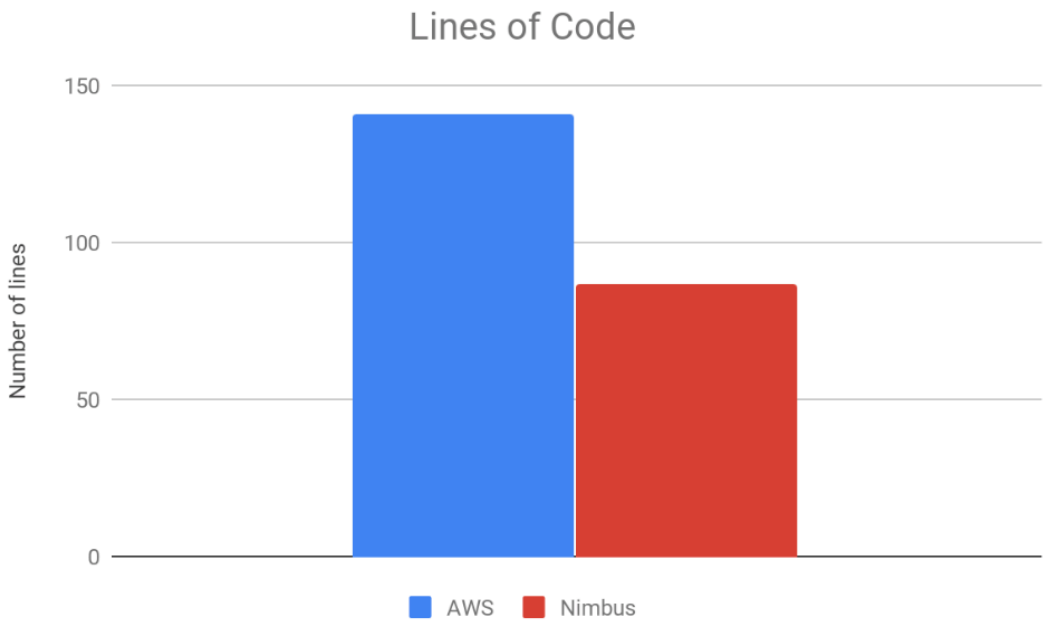
No local debugging / integration testing



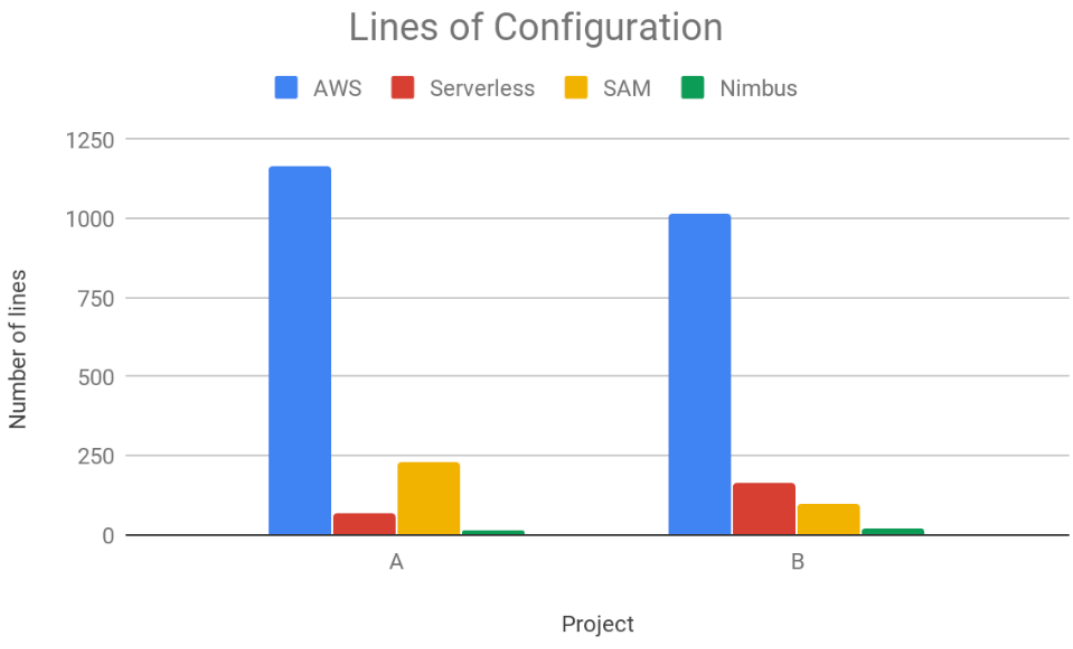
https://www.nimbusframework.com



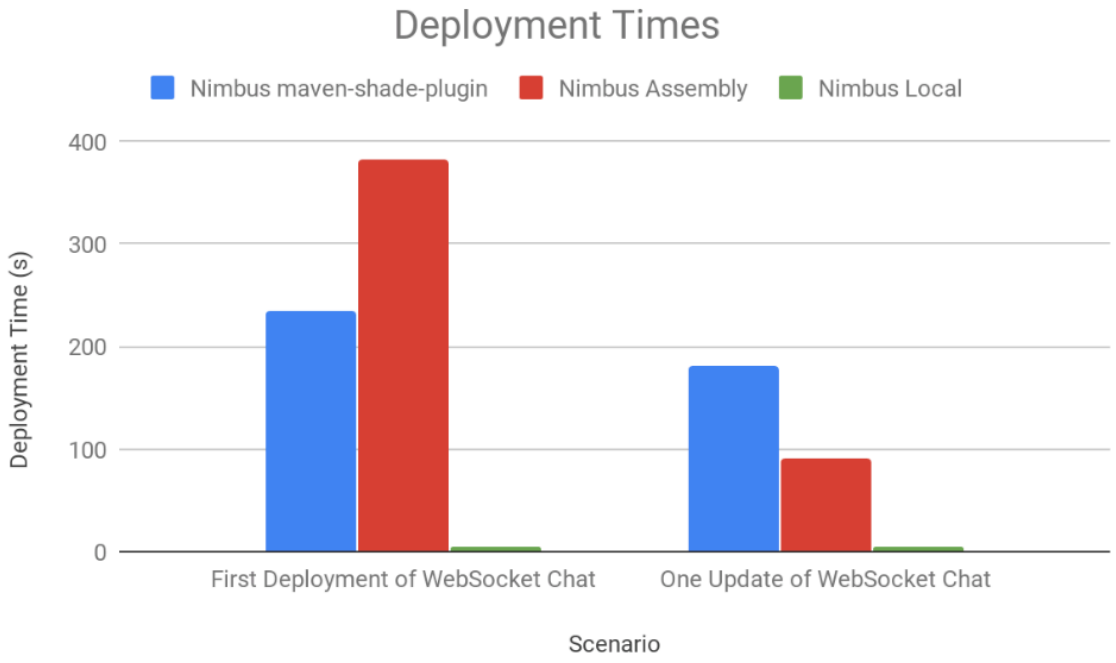
Comparisons



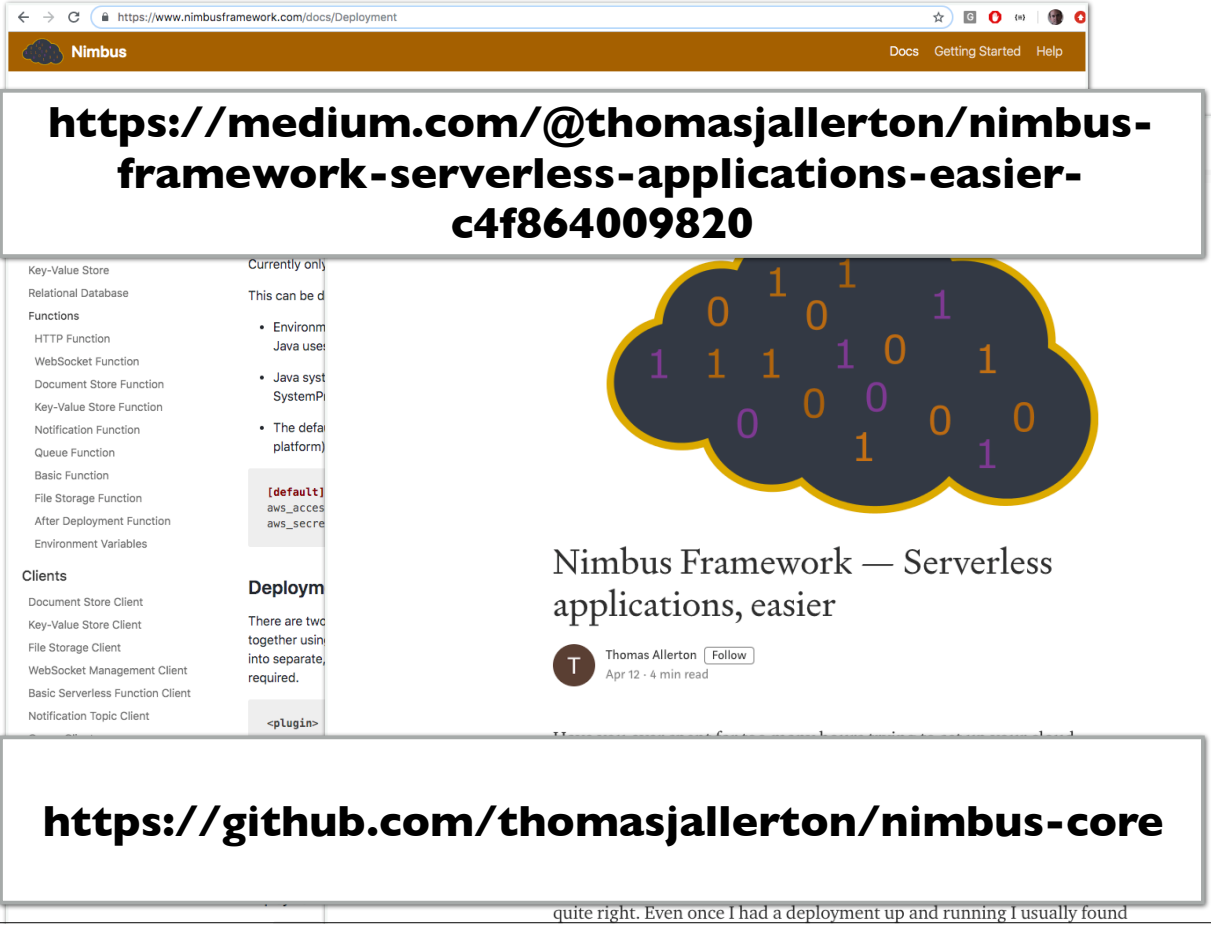
Comparisons



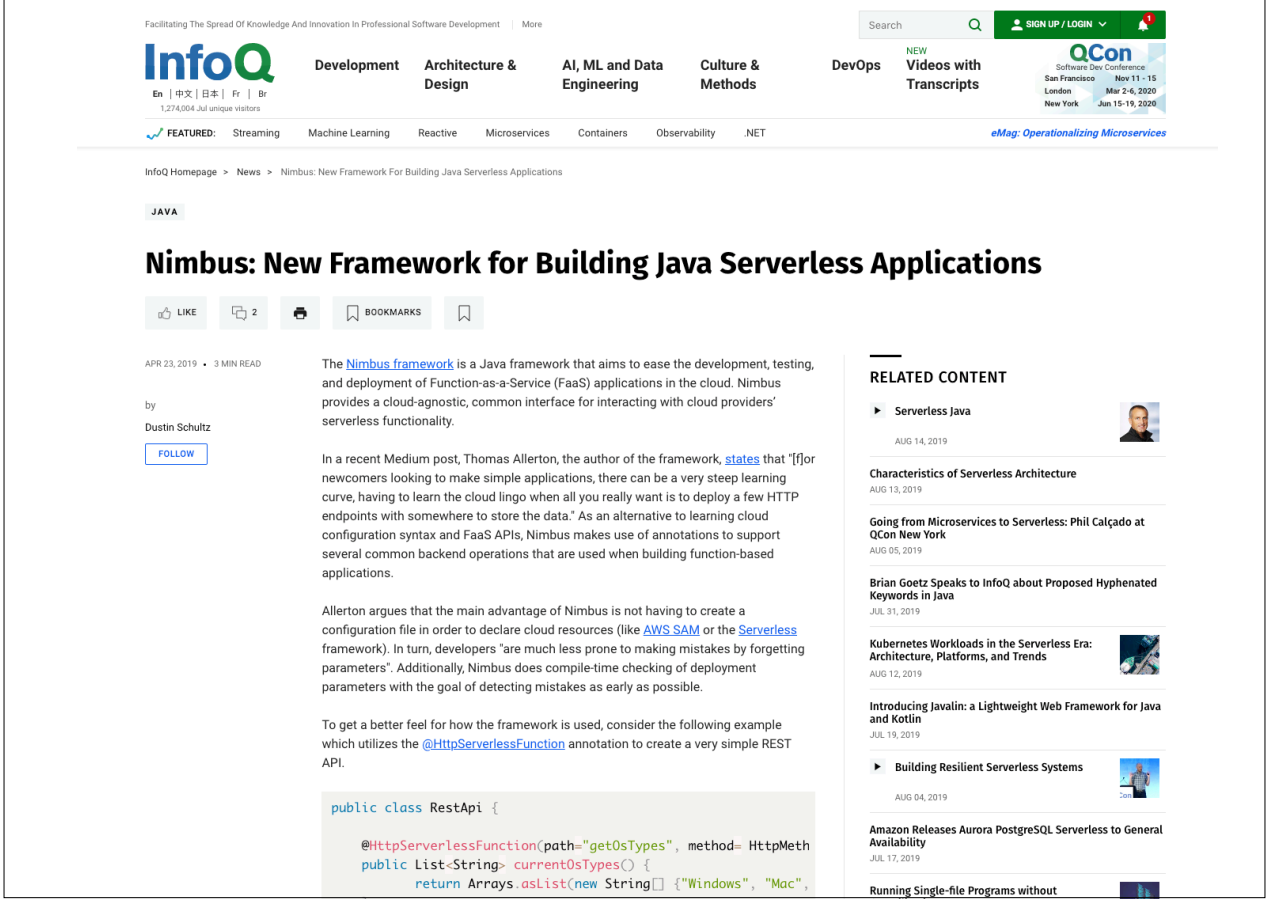
Comparisons



https://medium.com/@thomasjallerton/nimbus-framework-serverless-applications-easier-c4f864009820



https://github.com/thomasjallerton/nimbus-core



Facilitating The Spread Of Knowledge And Innovation In Professional Software Development | More

Search | SIGN UP / LOGIN

InfoQ

En | 中文 | 日本 | Fr | Br

1,274,004 Jul unique visitors

Development | Architecture & Design | AI, ML and Data Engineering | Culture & Methods | DevOps | NEW Videos with Transcripts

QCon Software Dev Conference
San Francisco Nov 11 - 15
London Mar 2-6, 2020
New York Jun 15-19, 2020

FEATURED: Streaming | Machine Learning | Reactive | Microservices | Containers | Observability | .NET

eMag: Operationalizing Microservices

InfoQ Homepage > News > Nimbus: New Framework For Building Java Serverless Applications

JAVA

Nimbus: New Framework for Building Java Serverless Applications

LIKE | 2 | BOOKMARKS

APR 23, 2019 • 3 MIN READ

by

Dustin Schultz

FOLLOW

The Nimbus framework is a Java framework that aims to ease the development, testing, and deployment of Function-as-a-Service (FaaS) applications in the cloud. Nimbus provides a cloud-agnostic, common interface for interacting with cloud providers' serverless functionality.

In a recent Medium post, Thomas Allerton, the author of the framework, states that "[f]or newcomers looking to make simple applications, there can be a very steep learning curve, having to learn the cloud lingo when all you really want is to deploy a few HTTP endpoints with somewhere to store the data." As an alternative to learning cloud configuration syntax and FaaS APIs, Nimbus makes use of annotations to support several common backend operations that are used when building function-based applications.

Allerton argues that the main advantage of Nimbus is not having to create a configuration file in order to declare cloud resources (like AWS SAM or the Serverless framework). In turn, developers "are much less prone to making mistakes by forgetting parameters". Additionally, Nimbus does compile-time checking of deployment parameters with the goal of detecting mistakes as early as possible.

To get a better feel for how the framework is used, consider the following example which utilizes the @HttpServerlessFunction annotation to create a very simple REST API.

```
public class RestApi {

    @HttpServerlessFunction(path="getOsTypes", method= HttpMeth
    public List<String> currentOsTypes() {
            return Arrays.asList(new String[] {"Windows", "Mac",
```

RELATED CONTENT

Serverless Java
AUG 14, 2019

Characteristics of Serverless Architecture
AUG 13, 2019

Going from Microservices to Serverless: Phil Calçado at QCon New York
AUG 05, 2019

Brian Goetz Speaks to InfoQ about Proposed Hyphenated Keywords in Java
JUL 31, 2019

Kubernetes Workloads in the Serverless Era: Architecture, Platforms, and Trends
AUG 12, 2019

Introducing Javalin: a Lightweight Web Framework for Java and Kotlin
JUL 19, 2019

Building Resilient Serverless Systems
AUG 04, 2019

Amazon Releases Aurora PostgreSQL Serverless to General Availability
JUL 17, 2019

Running Single-file Programs without

@rchatley #vss19

Questions and Suggestions?

Robert Chatley

Thomas Allerton

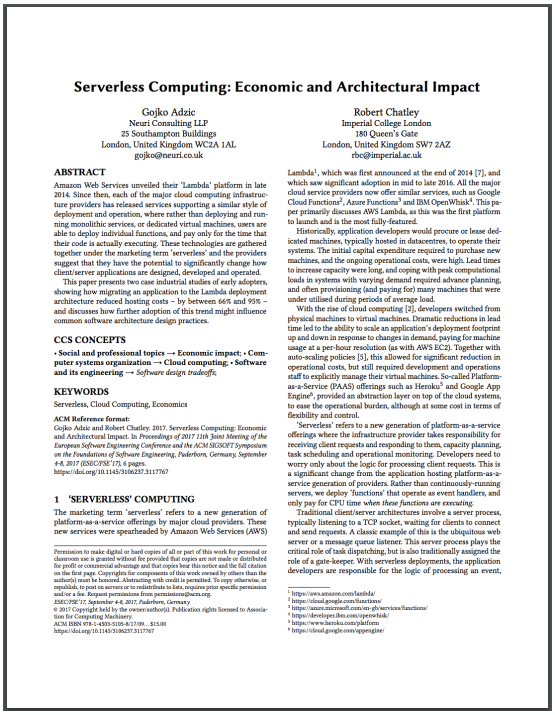
Imperial College London

RADON

http://radon-h2020.eu

Serverless Computing: Economic and Architectural Impact

Adzic and Chatley FSE'17



Serverless Computing: Economic and Architectural Impact

Gojko Adzic
Neuri Consulting LLP
25 Southampton Buildings
London, United Kingdom WC2A 1AL
gojko@neuri.co.uk

Robert Chatley
Imperial College London
180 Queen's Gate
London, United Kingdom SW7 2AZ
rbc@imperial.ac.uk

ABSTRACT

Amazon Web Services unveiled their 'Lambda' platform in late 2014. Since then, each of the major cloud computing infrastructure providers has released services supporting a similar style of deployment and operation, where rather than deploying and running monolithic services, or dedicated virtual machines, users are able to deploy individual functions, and pay only for the time that their code is actually executing. These technologies are gathered together under the marketing term 'serverless' and the providers suggest that they have the potential to significantly change how client/server applications are designed, developed and operated.

This paper presents two case industrial studies of early adopters, showing how migrating an application to the Lambda deployment architecture reduced hosting costs – by between 66% and 95% – and discusses how further adoption of this trend might influence common software architecture design practices.

CCS CONCEPTS

• **Social and professional topics** → **Economic impact**; • **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → *Software design tradeoffs*;

KEYWORDS

Serverless, Cloud Computing, Economics

ACM Reference format:
Gojko Adzic and Robert Chatley. 2017. Serverless Computing: Economic and Architectural Impact. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4-8, 2017 (ESEC/FSE'17),* 6 pages.
https://doi.org/10.1145/3106237.3117767

1 'SERVERLESS' COMPUTING

The marketing term 'serverless' refers to a new generation of platform-as-a-service offerings by major cloud providers. These new services were spearheaded by Amazon Web Services (AWS) Lambda[1], which was first announced at the end of 2014 [7], and which saw significant adoption in mid to late 2016. All the major cloud service providers now offer similar services, such as Google Cloud Functions[2], Azure Functions[3] and IBM OpenWhisk[4]. This paper primarily discusses AWS Lambda, as this was the first platform to launch and is the most fully-featured.

Historically, application developers would procure or lease dedicated machines, typically hosted in datacentres, to operate their systems. The initial capital expenditure required to purchase new machines, and the ongoing operational costs, were high. Lead times to increase capacity were long, and coping with peak computational loads in systems with varying demand required advance planning, and often provisioning (and paying for) many machines that were under utilised during periods of average load.

With the rise of cloud computing [2], developers switched from physical machines to virtual machines. Dramatic reductions in lead time led to the ability to scale an application's deployment footprint up and down in response to changes in demand, paying for machine usage at a per-hour resolution (as with AWS EC2). Together with auto-scaling policies [5], this allowed for significant reduction in operational costs, but still required development and operations staff to explicitly manage their virtual machines. So-called Platform-as-a-Service (PAAS) offerings such as Heroku[5] and Google App Engine[6], provided an abstraction layer on top of the cloud systems, to ease the operational burden, although at some cost in terms of flexibility and control.

'Serverless' refers to a new generation of platform-as-a-service offerings where the infrastructure provider takes responsibility for receiving client requests and responding to them, capacity planning, task scheduling and operational monitoring. Developers need to worry only about the logic for processing client requests. This is a significant change from the application hosting platform-as-a-service generation of providers. Rather than continuously-running servers, we deploy 'functions' that operate as event handlers, and only pay for CPU time *when these functions are executing.*

Traditional client/server architectures involve a server process, typically listening to a TCP socket, waiting for clients to connect and send requests. A classic example of this is the ubiquitous web server or a message queue listener. This server process plays the critical role of task dispatching, but is also traditionally assigned the role of a gate-keeper. With serverless deployments, the application developers are responsible for the logic of processing an event,

ESEC/FSE'17, September 4-8, 2017, Paderborn, Germany

ACM ISBN 978-1-4503-5105-8/17/09…$15.00
https://doi.org/10.1145/3106237.3117767

[1] https://aws.amazon.com/lambda/
[2] https://cloud.google.com/functions/
[3] https://azure.microsoft.com/en-gb/services/functions/
[4] https://developer.ibm.com/openwhisk/
[5] https://www.heroku.com/platform
[6] https://cloud.google.com/appengine/

Continuous Performance Testing in Virtual Time

Chatley, Field and Wei CSE/QUDOS'19

Continuous Performance Testing in Virtual Time

Robert Chatley
Department of Computing
180 Queen's Gate
London, SW7 2AZ, United Kingdom
rbc@imperial.ac.uk

Tony Field
Department of Computing
180 Queen's Gate
London, SW7 2AZ, United Kingdom
ajf@imperial.ac.uk

David Wei
Department of Computing
180 Queen's Gate
London, SW7 2AZ, United Kingdom
dw512@imperial.ac.uk

***Abstract*—We introduce the notion of performance unit testing which allows developers to explore performance characteristics and detect potential performance problems continuously throughout the development of a software system. Our ideas are embodied in *PerfMock*, which extends a well-established object mocking framework so that each mock object can be configured with a performance model for predicting the time taken to process each message it receives. *PerfMock* executes tests in *virtual* time. This allows performance to be evaluated much more quickly than running a full system performance test, making it possible to test performance continuously, as part of a unit test suite. We demonstrate the core features of *PerfMock* and show how it can be used to support a process of iterative refinement, whereby models can be improved when more about the actual performance of the objects being mocked becomes known, e.g. by building models from production data. We show that even very simple performance models used early on in the development process can provide useful information for estimating both absolute execution times and the effects of changes in functionality and/or design. The iterative approach we support has the pleasing property that as the system evolves, more decisions are made and more data is collected meaning that we can refine our models, and predicted and actual performance gradually converge.**

I. Introduction

The widespread adoption of agile methods [1] and continuous delivery [2] of software has resulted in development processes that are dependent on the use of rapid feedback from automated testing. In order to obtain fast feedback, developers often concentrate on testing small units in isolation, typically with pure unit tests [3]. This sort of testing has proved valuable in practice, but it cannot tell the developer everything. For example, unit tests do not indicate whether the system works as a whole to achieve a business goal, nor are they capable of evaluating the user experience. Of particular interest to this paper is the fact that they do not address performance.

Performance testing is typically done later on in development once a complete version of the software system can be deployed, instead of a primary concern that drives the software development process [4]. Performance issues are therefore usually not exposed until the system is integrated and tested as a whole. Resolving performance problems at this stage can be expensive, as it may involve redesigning parts of the system, rewriting code or allocating more computing resources to certain components to match requirements [5]. Performance tests are also typically slow or inconvenient to run, which is at odds with the fast feedback loops associated with test-driven development (TDD), i.e. the 'red, green, refactor' [6] loop, gaining confidence of correctness after every change.

The objective of this paper is to extend existing TDD techniques so that performance-related properties can be continuously verified throughout the software development process. For example, we may wish to establish that a class A will meet its required performance characteristics given that its collaborator B has a performance profile that matches X. The key idea is to capture X, using a *performance model*.

Our approach builds on the well-established idea of using *mock objects* to conduct unit testing in isolation [7]. Mock objects are used to replace the real collaborators of an object under test with implementations that serve only to support the test. Mock objects can be configured to behave in particular ways to simulate different scenarios, and can also be used to verify that the expected messages are exchanged between the various collaborators in a given test scenario. We extend this technique to allow mock objects to be configured with embedded performance models that are responsible for predicting the time that the object being mocked will take to respond to a message. This enables *performance unit tests* to be written that make assertions about performance as well as behaviour.

A performance model is any piece of code that is capable of estimating a time delay, e.g. by straightforward distribution sampling, the solution of a mathematical model such as a Markov Process or product-form queueing network, or by running a discrete-event simulation alongside the unit under test. The choice of model is up to the developer and our approach supports arbitrary models types, including the above. Whatever model is chosen from the beginning, the intention is that it can be refined iteratively as more becomes known about the actual behaviour of the object being modelled, e.g. through ongoing integration testing or data from production deployment – see Section III-A.

A key point is that performance models work entirely in *virtual* time, which means that performance estimates can be produced without having to wait for the passage of real time. This leads to fast turnaround times, which is one of the key requirements of effective automated testing, and enables large suites of performance tests to be included in a pre-commit build or a continuous delivery pipeline, without significantly increasing the build time.

The ability to do early-stage continuous performance testing is a realisation of the software performance engineer-