jakarta ee, microprofile and clouds

adam-bien.com

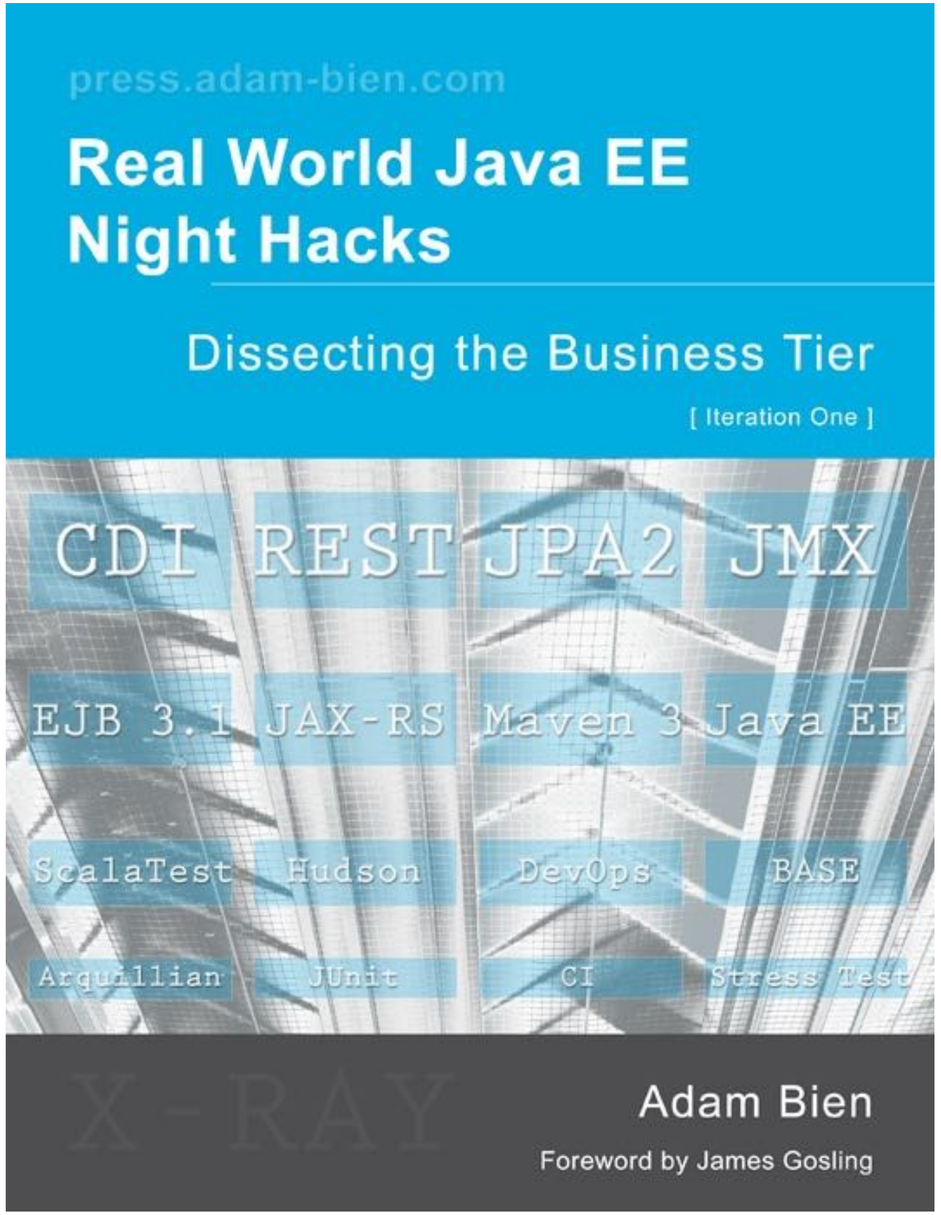
"It's not work if you like it"
…so I never worked. #java

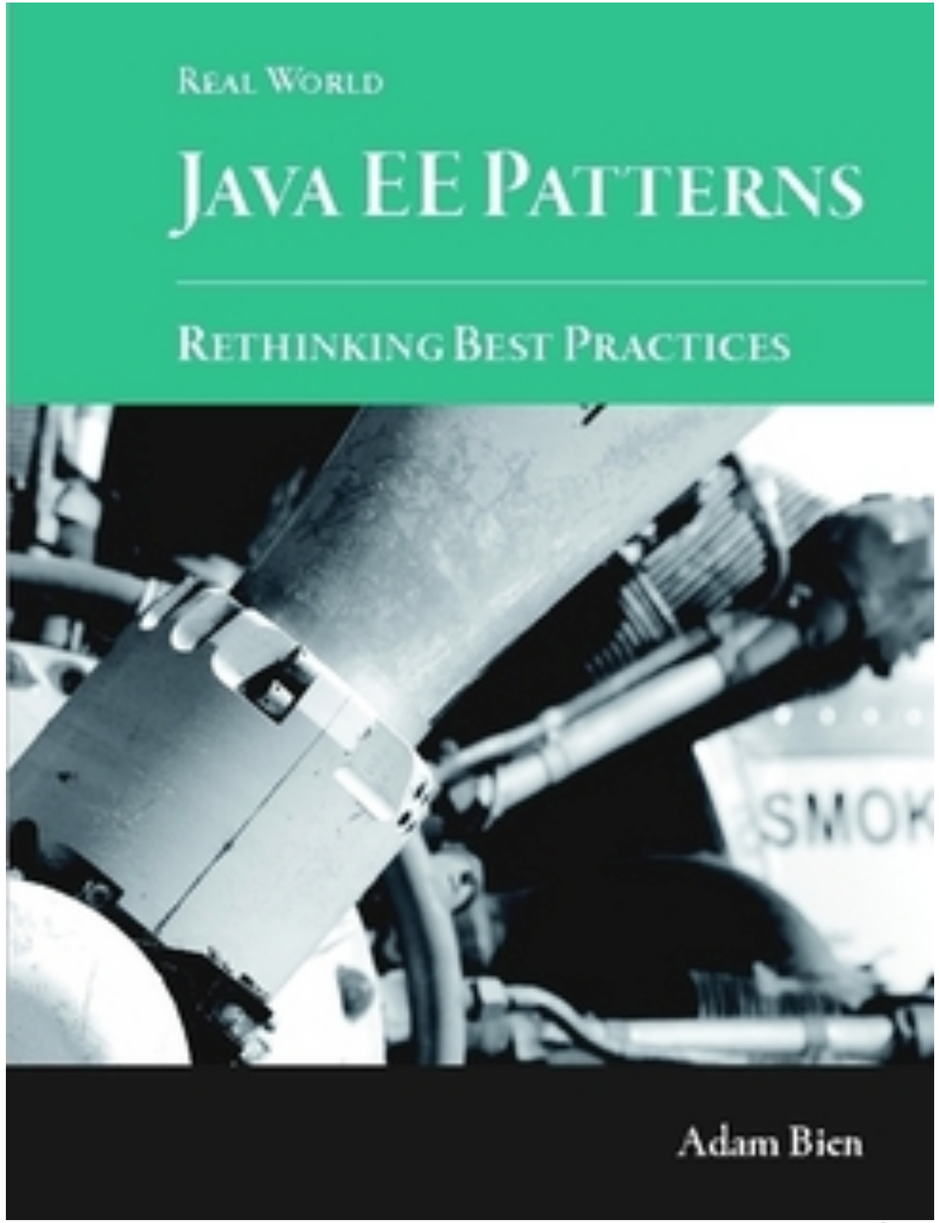
adam-bien.com

learn once,
never migrate

adambien.blog
airhacks.io
airhacks.tv
airhacks.news

airhacks.fm





adam-bien.com

airhacks.TV

adam-bien.com

Munich (MUC) Airport Web Workshops for Java Developers, Summer 2020:

WebComponents Kickstarter, June 30th, 2020

partially also available as: [Streaming / Download Edition]

Building Apps with WebComponents, July 1st, 2020

available as: [Streaming / Download Edition]

Munich (MUC) Airport Workshops, Winter 2019:

Jakarta EE / MicroProfile Microservices, December 10th, 2019

also available as: [Streaming / Download Edition]

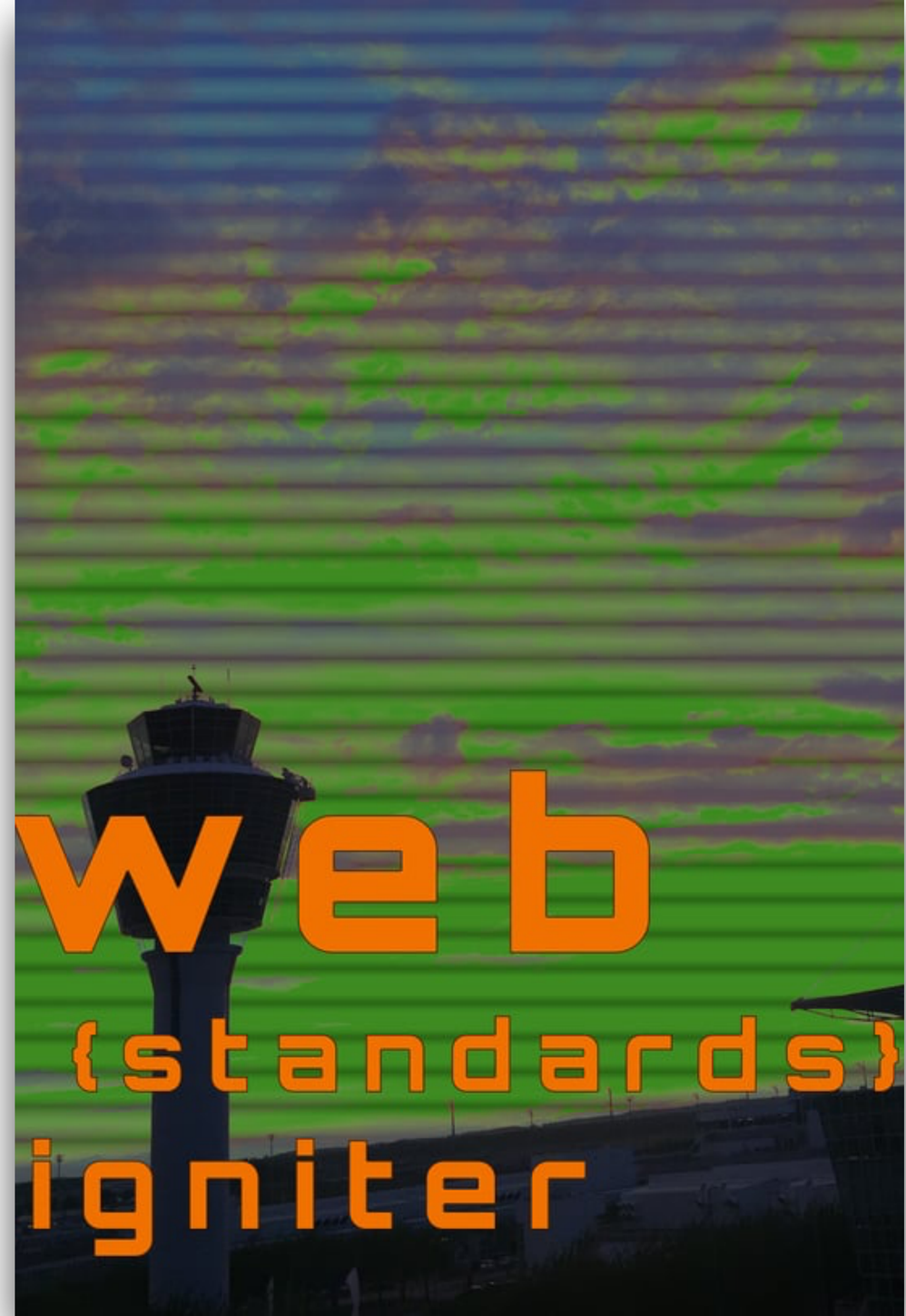
Jakarta EE, Microprofile and Clouds, December 11th, 2019

Beyond Boring Jakarta EE and JDK 12--From NoSQL, over Reactive to Fibers, December 12th, 2019

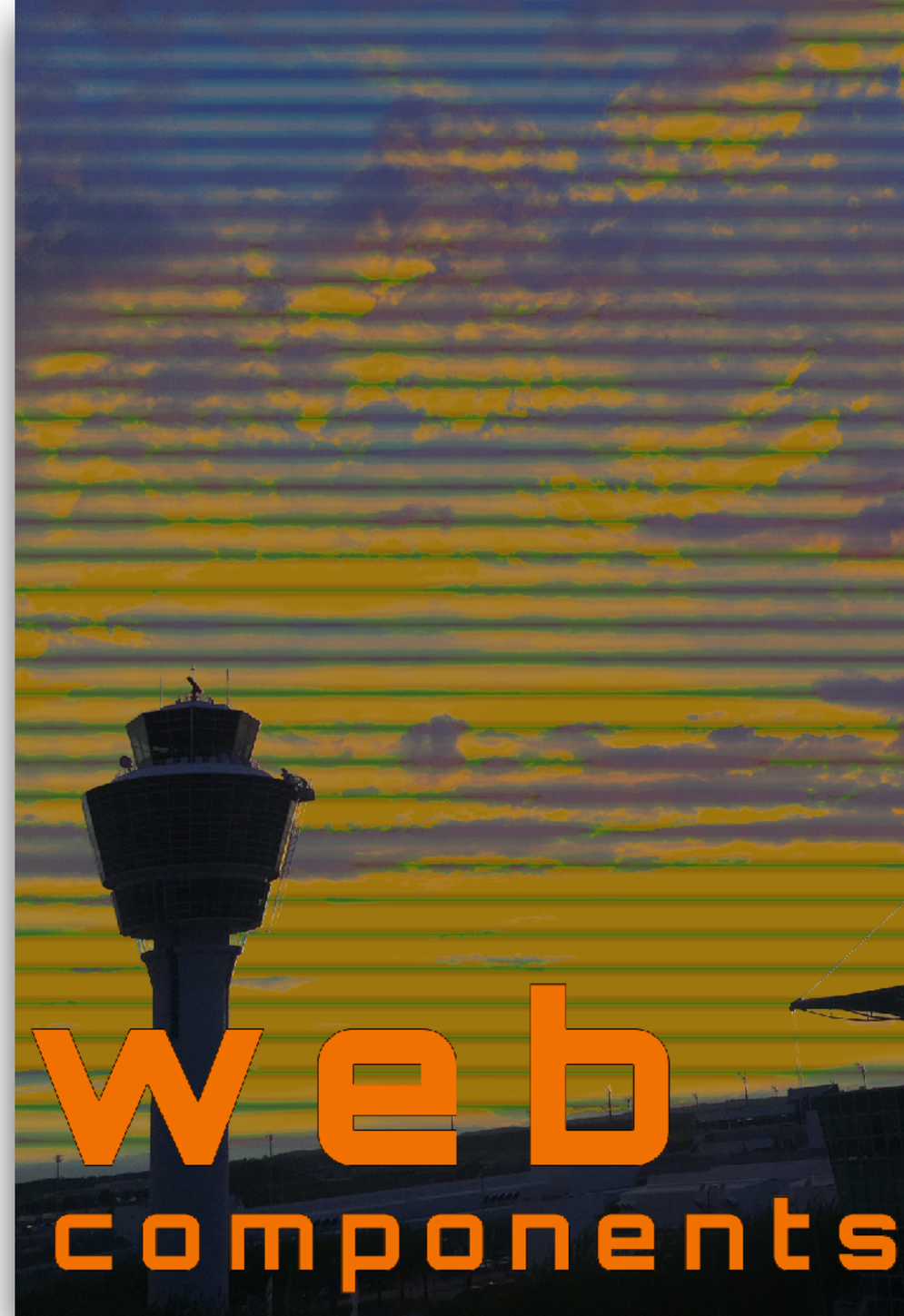
NEW:

Streaming Architectures, December 13th, 2019

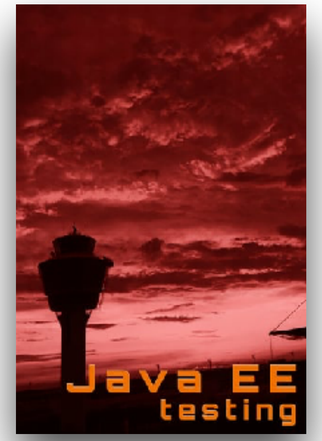
http://webstandards.training

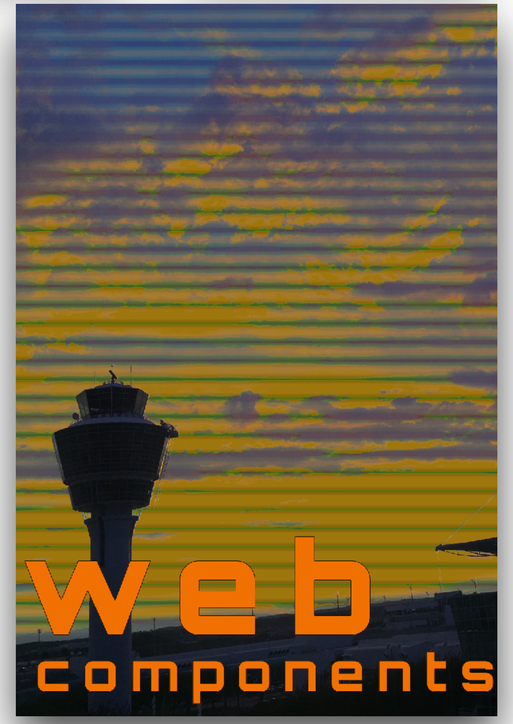


http://webcomponents.training



http://effectiveweb.training

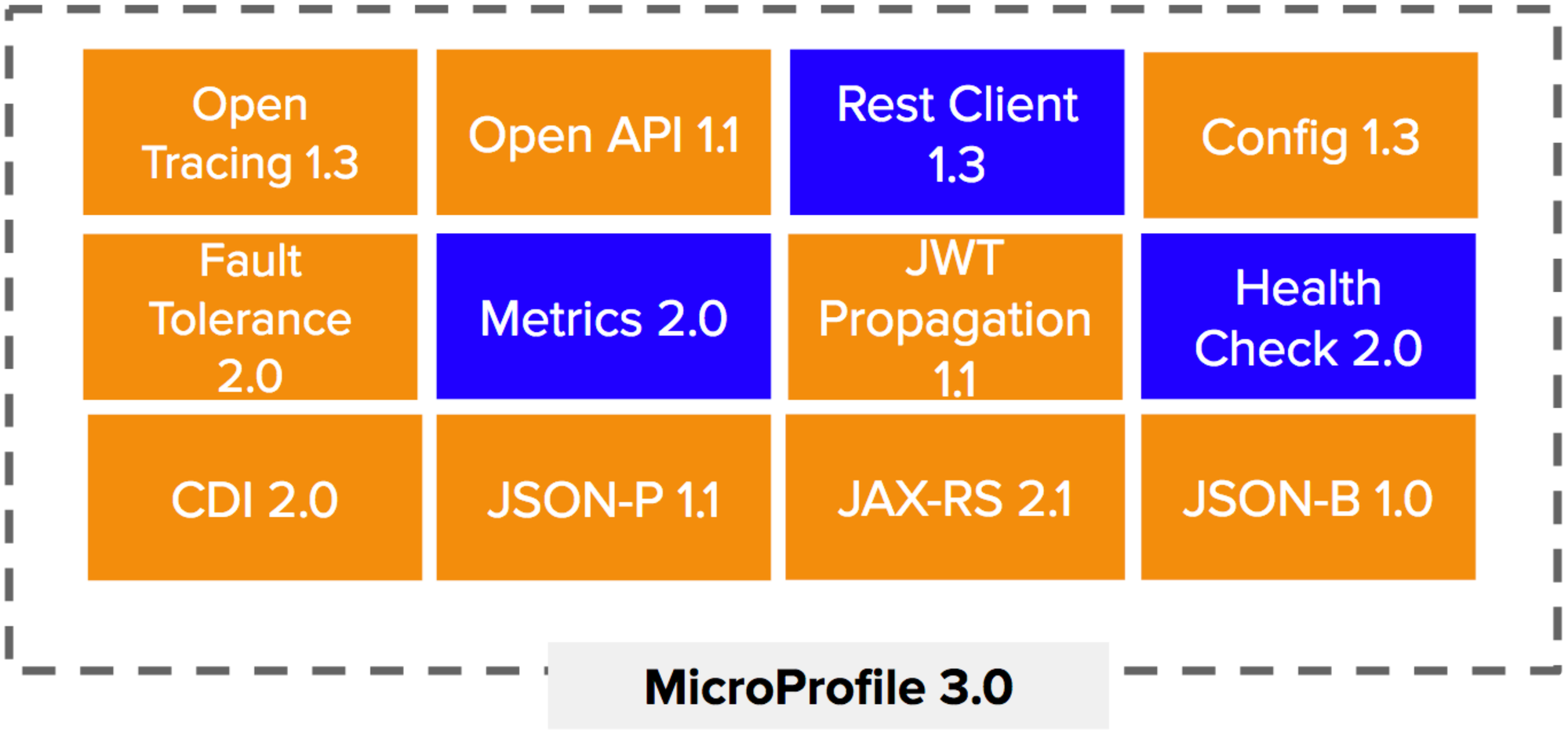




airhacks.io

javaeemicro.services

MicroProfile 3.0


Open Tracing 1.3
Open API 1.1
Rest Client 1.3
Config 1.3
Fault Tolerance 2.0
Metrics 2.0
JWT Propagation 1.1
Health Check 2.0
CDI 2.0
JSON-P 1.1
JAX-RS 2.1
JSON-B 1.0
MicroProfile 3.0
= New
= Updated
= No change from last release (MicroProfile 2.2)

What is cloud native?

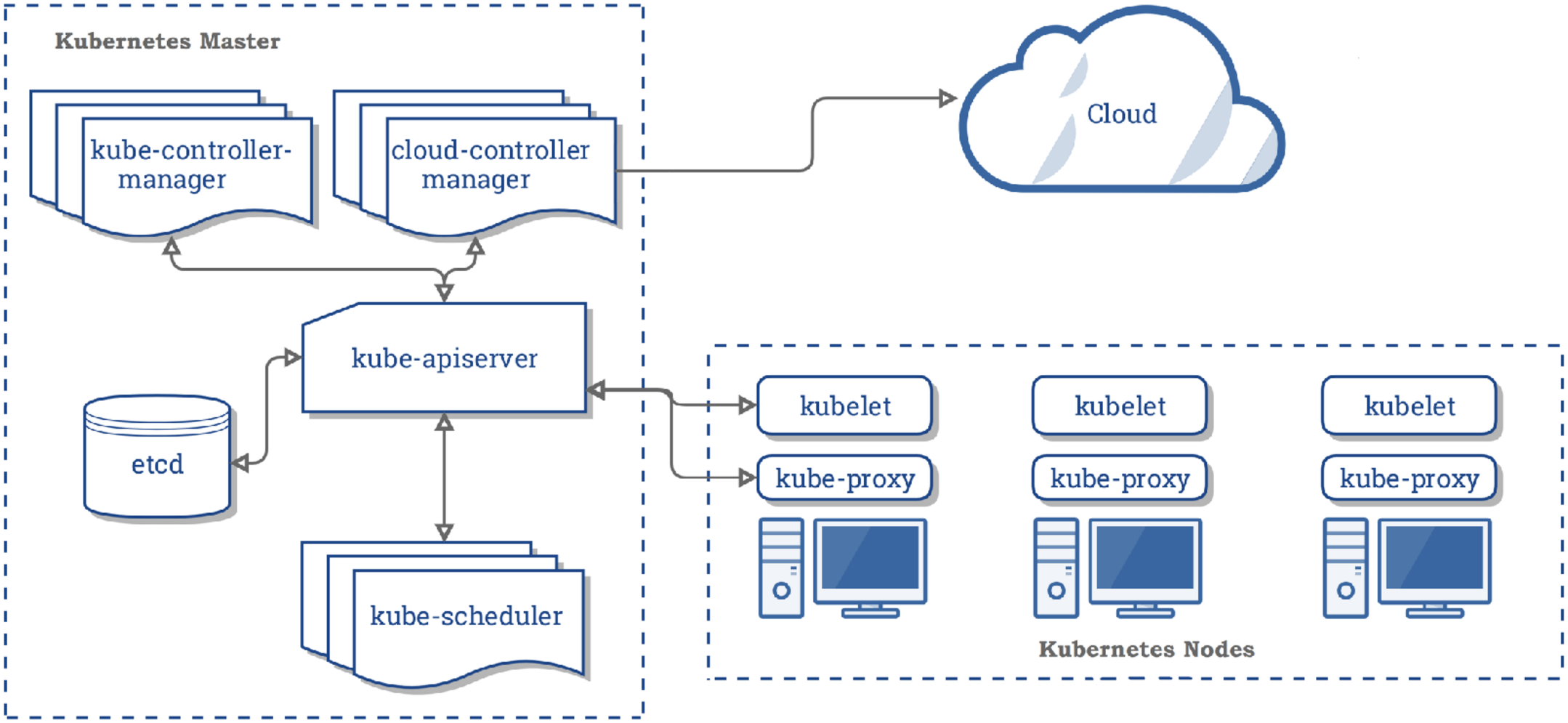
Docker

Registries

Clouds

public / private / hybrid

Kubernetes



https://kubernetes.io/docs/concepts/overview/components/

“Patterns”

Timeouts

Bulkheads

Circuit Breaker

Retry

Fallback

12factor.net

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Serverless

Serverless computing, also known as function as a service (FaaS), is a cloud computing code execution model in which the cloud provider fully manages starting and stopping of a function's container platform as a service (PaaS) as necessary to serve requests, and requests are billed by an abstract measure of the resources required to satisfy the request, rather than per virtual machine, per hour.

Despite the name, it does not actually involve running code without servers.[1] The name "serverless computing" is used because the business or person that owns the system does not have to purchase, rent or provision servers or virtual machines for the back-end code to run on.

Serverless code can be used in conjunction with code written in traditional server style, such as microservices. For example, part of a web application could be written as microservices and another part could be written as serverless code. Alternatively, an application could be written that uses no provisioned servers at all, being completely serverless

Reactive Programming

adam-bien.com

Reactive programming

From Wikipedia, the free encyclopedia

In computing, **reactive programming** is a programming paradigm oriented around data flows and the propagation of change. This means that it should be possible to express static or dynamic data flows with ease in the programming languages used, and that the underlying execution model will automatically propagate changes through the data flow.

For example, in an imperative programming setting, $a := b + c$ would mean that a is being assigned the result of $b + c$ in the instant the expression is evaluated. Later, the values of b and c can be changed with no effect on the value of a.

In reactive programming, the value of a would be automatically updated based on the new values.

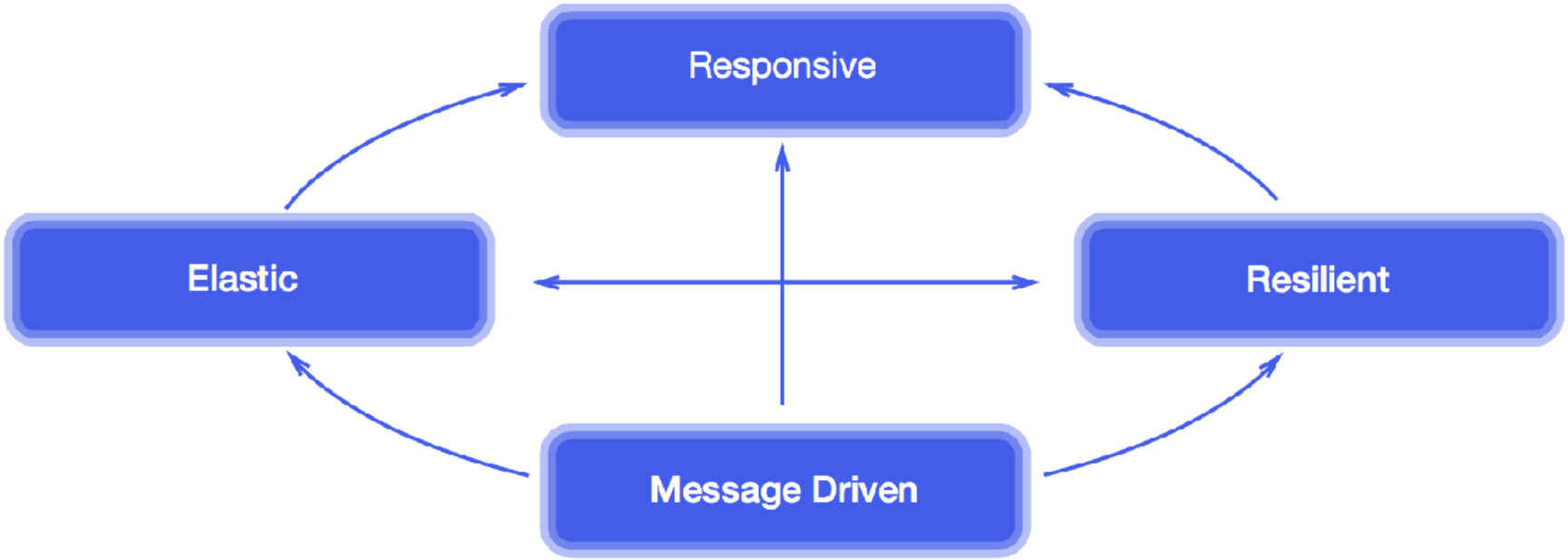
A modern spreadsheet program is an example of reactive programming. Spreadsheet cells can contain literal values, or formulas such as "=B1+C1" that are evaluated based on other cells. Whenever the value of the other cells change, the value of the formula is automatically updated.

Another example is a hardware description language such as Verilog. In this case reactive programming allows us to model changes as they propagate through a circuit.

Reactive programming has foremost been proposed as a way to simplify the creation of interactive user interfaces, animations in real time systems, but is essentially a general programming paradigm.

http://en.wikipedia.org/wiki/Reactive_programming

Reactive Manifesto



http://www.reactivemanifesto.org

Responsive

The system responds in a timely manner if at all possible. Responsiveness is the cornerstone of usability and utility, but more than that, responsiveness means that problems may be detected quickly and dealt with effectively. Responsive systems focus on providing rapid and consistent response times, establishing reliable upper bounds so they deliver a consistent quality of service. This consistent behaviour in turn simplifies error handling, builds end user confidence, and encourages further interaction.

Resilient

The system stays responsive in the face of failure. This applies not only to highly-available, mission critical systems — any system that is not resilient will be unresponsive after a failure. Resilience is achieved by replication, containment, isolation and delegation. Failures are contained within each component, isolating components from each other and thereby ensuring that parts of the system can fail and recover without compromising the system as a whole. Recovery of each component is delegated to another (external) component and high-availability is ensured by replication where necessary. The client of a component is not burdened with handling its failures.

Elastic

The system stays responsive under varying workload. Reactive Systems can react to changes in the input rate by increasing or decreasing the resources allocated to service these inputs. This implies designs that have no contention points or central bottlenecks, resulting in the ability to shard or replicate components and distribute inputs among them. Reactive Systems support predictive, as well as Reactive, scaling algorithms by providing relevant live performance measures. They achieve elasticity in a cost-effective way on commodity hardware and software platforms.

Message Driven

Reactive Systems rely on asynchronous message-passing to establish a boundary between components that ensures loose coupling, isolation, location transparency, and provides the means to delegate errors as messages. Employing explicit message-passing enables load management, elasticity, and flow control by shaping and monitoring the message queues in the system and applying back-pressure when necessary. Location transparent messaging as a means of communication makes it possible for the management of failure to work with the same constructs and semantics across a cluster or within a single host. Non-blocking communication allows recipients to only consume resources while active, leading to less system overhead.

Munich (MUC) Airport Web Workshops for Java Developers, Summer 2020:

WebComponents Kickstarter, June 30th, 2020

partially also available as: [Streaming / Download Edition]

Building Apps with WebComponents, July 1st, 2020

available as: [Streaming / Download Edition]

Munich (MUC) Airport Workshops, Winter 2019:

Jakarta EE / MicroProfile Microservices, December 10th, 2019

also available as: [Streaming / Download Edition]

Jakarta EE, Microprofile and Clouds, December 11th, 2019

Beyond Boring Jakarta EE and JDK 12--From NoSQL, over Reactive to Fibers, December 12th, 2019

NEW:

Streaming Architectures, December 13th, 2019

Thank You!

blog.adam-bien.com
twitter.com/AdamBien