Domain Driven Design in Cloud Native Environments

ISAQB Architecture Gathering, Nov 17, 2022

Tobias Goeschel (he/him)
Sr Solutions Architect, FSI
Amazon Web Services

tgo@amazon.de
@w3ltraumpirat

© 2022, Amazon Web Services, Inc. or its affiliates.

Say hello to Harry.

Harry - gefangen in der Zeit

© 2022, Amazon Web Services, Inc. or its affiliates.

Say hello to Harry.

© 2022, Amazon Web Services, Inc. or its affiliates.

Say hello to Harry.

Say hello to Harry.

- German learning course, realized by freiwerk-b for Deutsche Welle
- 100 episodes / 4 games per ep. / 20 different game types
- 30 languages (incl. Arabic/right-to-left)
- Vocabulary trainer
- Fully editable / drag and drop for native speakers / editors

© 2022, Amazon Web Services, Inc. or its affiliates.

Say hello to Harry.

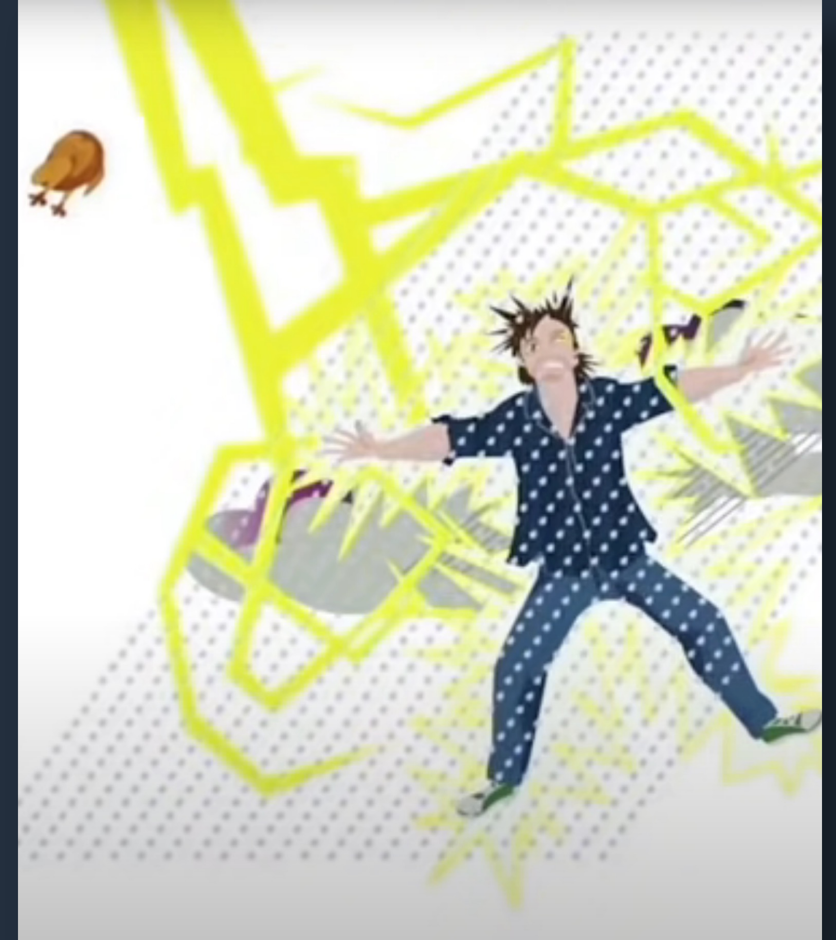
- Originally planned 2009/10
- Flash (ActionScript 3), Java, Jboss 4, MySQL
- Accessible version in HTML5 (Apache Wicket)
- CRUD based
- Domain specific language, lots of generated code
- 4 devs, but then it was only me
- This was going to be my big breakthrough

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Say hello to Harry.

- One year in: Editors hate it.
  - Full redesign of the editing app.
- Two years in: It doesn't scale.
  - Full refactor/rewrite with DDD
  - Invented an extension to CSS to enable content positioning
- Project eventually finished in 2014
- Burnout, 45K debt, got a permanent job



© 2022, Amazon Web Services, Inc. or its affiliates.

Say hello to Harry.

I learned a few things:

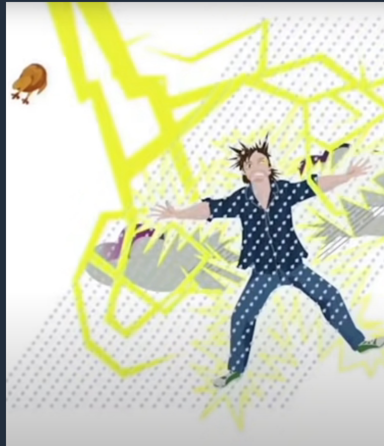
- I don’t actually need other people to screw up the code. I can do that all by myself.
- The who, why and how of software development is ~~almost~~ always more important than the tooling.
- I quit the generators. And decided to become an expert for Software Crafting and DDD instead.

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

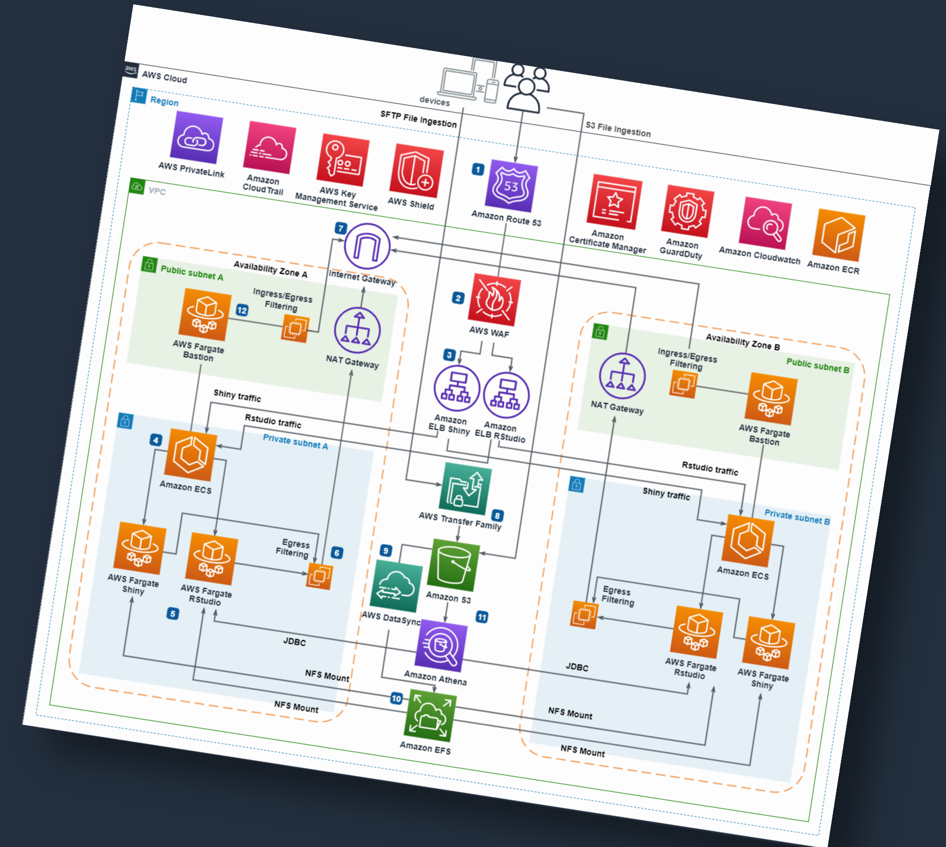
Why am I telling you all this?

Well, ultimately, it's how I ended up here.



© 2022, Amazon Web Services, Inc. or its affiliates.

A DDD practitioner walks into a bar…



© 2022, Amazon Web Services, Inc. or its affiliates.

A DDD practitioner walks into a bar…

- ”Architecture” translates to technical artifacts/products
- Everything is distributed
- Everything is an API
- Everything is billed by consumption
- Everything is ideally a managed service
- Everything is serverless
- Everything is automated
- Everything is secure by default

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

A DDD practitioner walks into a bar…

- ”Architecture” translates to t
- Everything is distributed
- Everything is an API
- Everything is billed by cons
- Everything is ideally a man
- Everything is serverless
- Everything is automated
- Everything is secure by default

© 2022, Amazon Web Services, Inc. or its affiliates.

Let’s talk about boundaries

© 2022, Amazon Web Services, Inc. or its affiliates.

- Essential building blocks of a system. They influence decisions on many levels, and have **social, technological, and political** dimensions.
- Should always be defined by (ubiquitous) **language**.
- **Assumption: All this is also true for the cloud.**

Bounded Contexts

- A BCs is one possible technical realization of a (sub-) domain.
- A (sub-) domain can be implemented by one or more BCs, but one BC should not belong to more than one (sub-) domain.
  - **Caution:** Often not true in legacy / generic / supporting domains!
- One team can own more than one BC, but a BC should not be owned by more than one team

aws
© 2022, Amazon Web Services, Inc. or its affiliates.

Bounded Contexts

- Eric Evans/Vaughn Vernon: Implement BCs as modules (packages)
- Microservices: Implement one µS per BC
- Serverless:

© 2022, Amazon Web Services, Inc. or its affiliates.

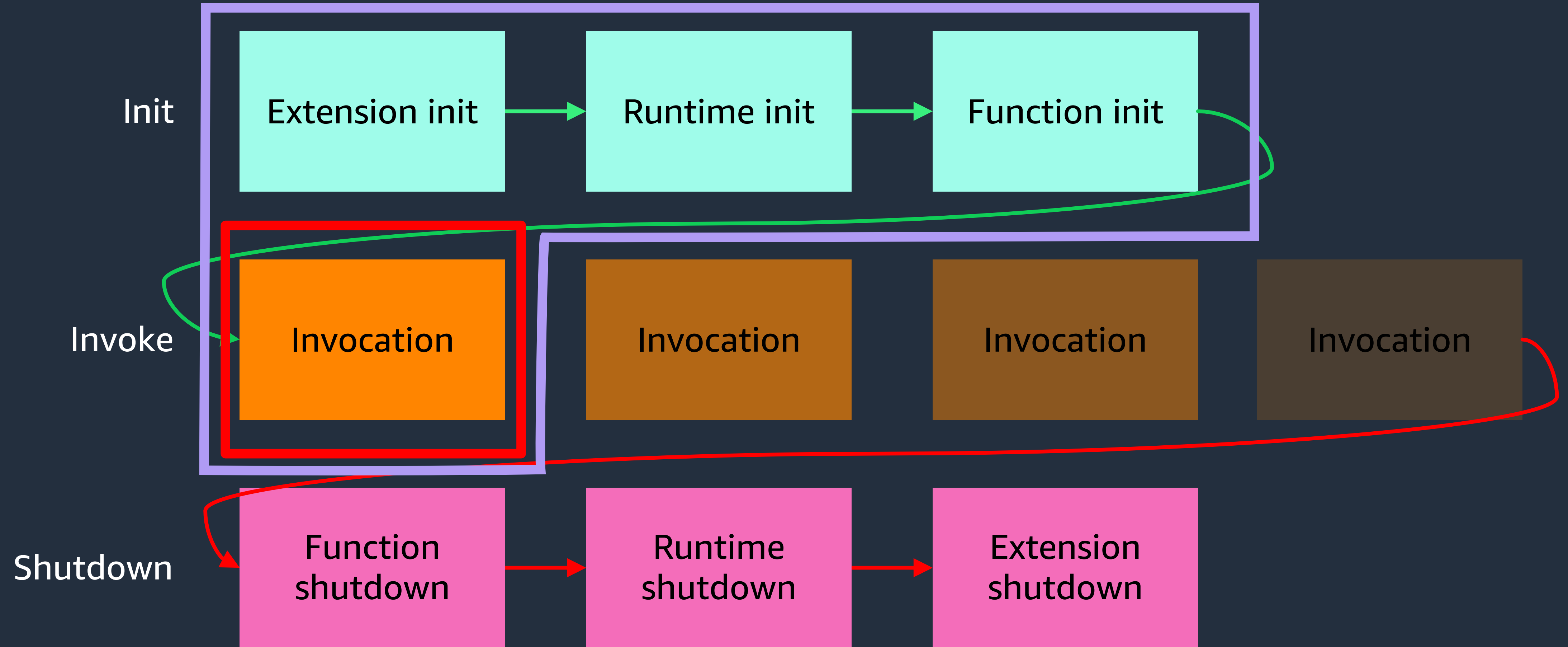
Bounded Contexts

- Serverless components are billed by **consumption**
  - They should not be running all the time!
  - They must be **stateless**
  - Consider **startup times** / latency

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Anatomy of a serverless function



© 2022, Amazon Web Services, Inc. or its affiliates.

Bounded Contexts

- Serverless components are billed by **consumption**
  - They should not be running all the time!
  - They must be **stateless**
  - Consider **startup times** / latency
  - They should **scale automatically** – parallelism!
  - They should be **as small as possible** (Single Responsibility Principle)
    - **Caution:** Serverless functions are also billed by **# of executions**!

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Bounded Contexts

- With all these small, auto-scaled, volatile components, how do we implement context boundaries?
  - Options:
    - One VPC per BC?
      - Not always allowed, and might incur extra cost
    - One account **per team** OR one account **per BC**?
      - Use **Landing Zone / Account Factory** to self-service
        - **Caution: Platforms** are also a dependency!

© 2022, Amazon Web Services, Inc. or its affiliates.

Bounded Contexts

But… What about Kubernetes?

- One BC per **deployment**?
- One BC OR one team per **namespace**?
- One team per **cluster**?
  - Remember: Platforms are a dependency!

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Let’s talk about boundaries (2)

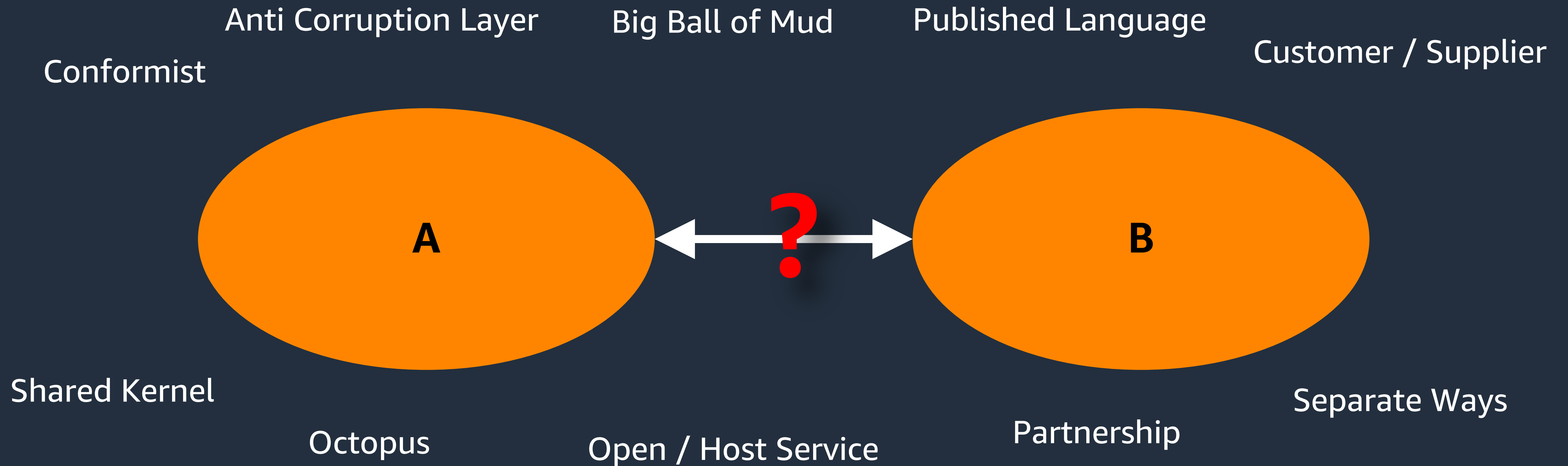
aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Integrations

Bounded contexts (usually) cannot exist in isolation.

Integrations



© 2022, Amazon Web Services, Inc. or its affiliates.

Integrations

Bounded contexts (usually) cannot exist in isolation.

- Be aware that integrations are equivalent to **contracts**.
- A contract regulates both acceptable **forms of communication**, and the **social dynamics** of the **team boundary**
- Use **context maps** to figure out the appropriate kind of integration/ nature of contract

Integrations

- Integrations can be implemented as **public interfaces** (e.g. REST, GraphQL or RPC) ,or via Messaging / **Domain Events.**
- Use **OpenAPI** or similar formats to make contracts explicit.
- Use **Consumer Driven Contract Tests** for documentation, and to ensure integration correctness.
- This is a high-level variation of the **Ports and Adapters** pattern (aka **Hexagonal Architecture**).

aws
© 2022, Amazon Web Services, Inc. or its affiliates.

Let’s talk about boundaries (3)

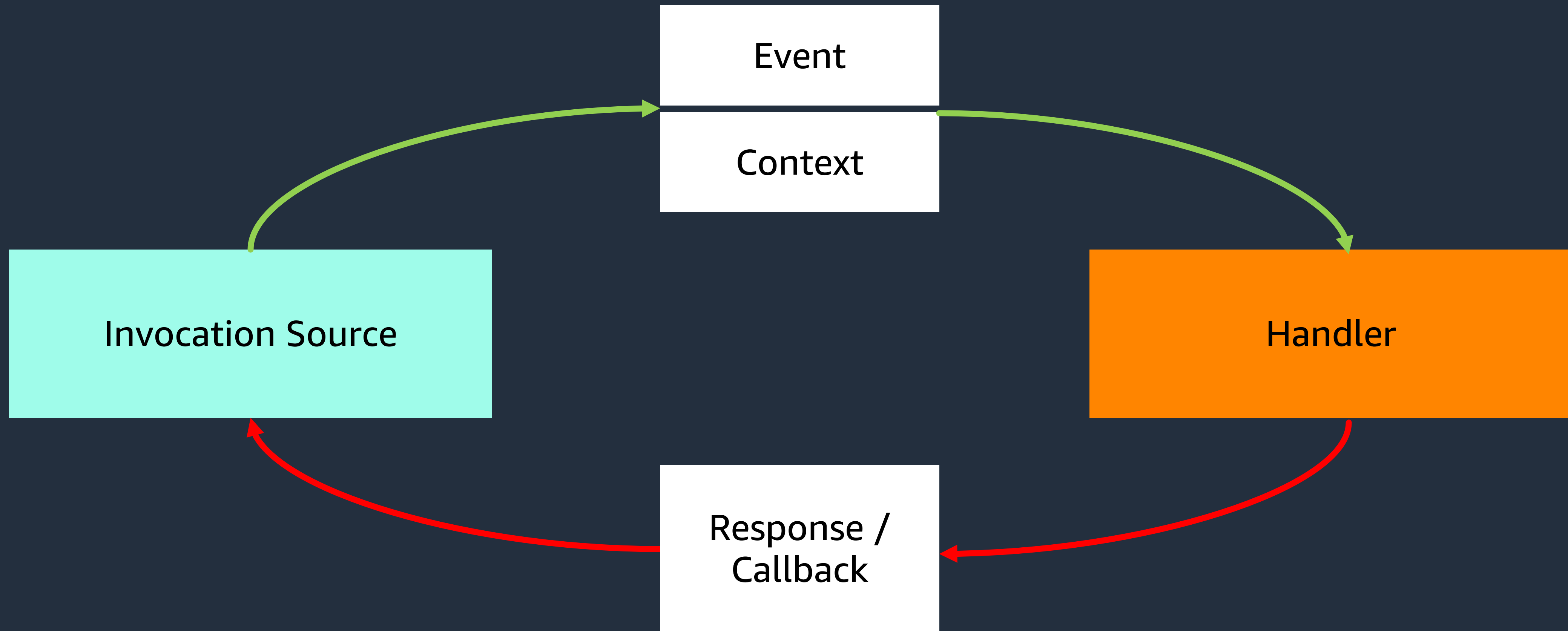
© 2022, Amazon Web Services, Inc. or its affiliates.

Modules

Can small components have even smaller parts? Of course, they can!

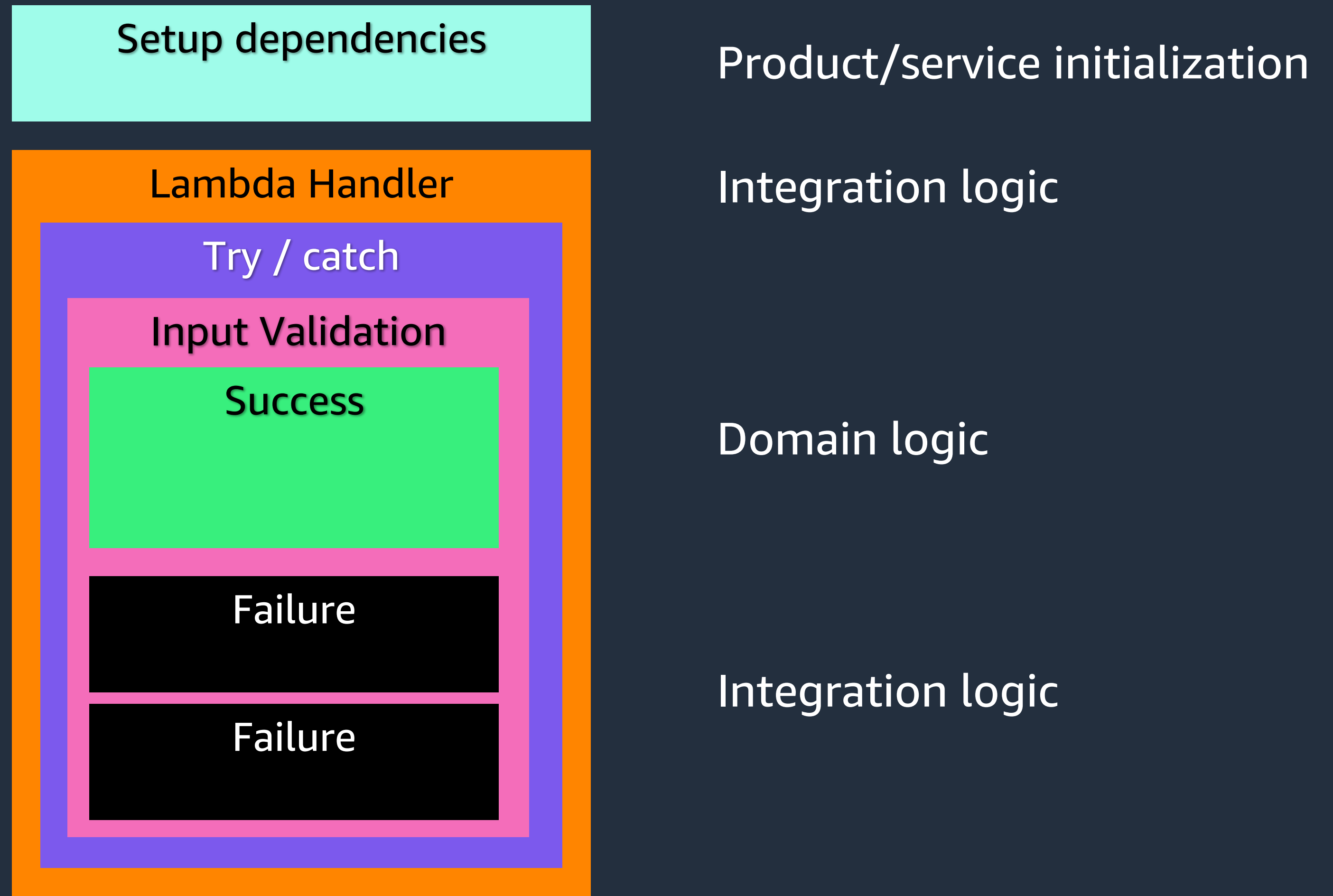
- Serverless functions can have **layers**.
- They work much like **package dependencies**.
- Caution: These are hard, binary dependencies! This means **coupling**.

Anatomy of a serverless function



© 2022, Amazon Web Services, Inc. or its affiliates.

Anatomy of a serverless function



© 2022, Amazon Web Services, Inc. or its affiliates.

Modules

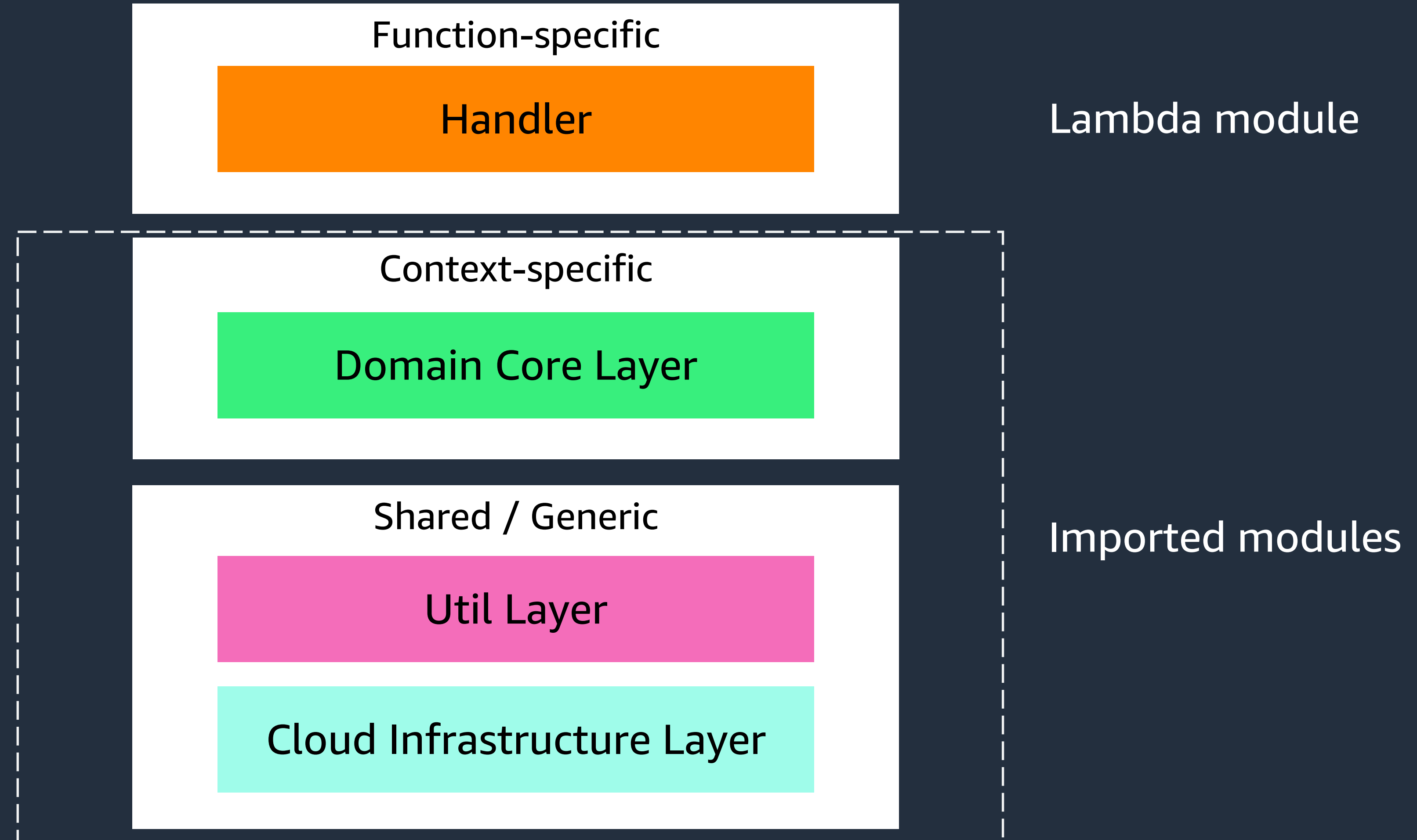
- Lamba/function (application) code should
  - wire **dependencies**
  - take care of **event and error handling**
  - map results to appropriate **response outputs**

  But always call validation and business logic from imported modules

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Anatomy of a serverless function



© 2022, Amazon Web Services, Inc. or its affiliates.

Modules

- Create a **separate layer** for your **domain core**
  - Use **Hexagonal Architecture** to keep it testable, dependency-free, and well-encapsulated
- Business logic must be shared **only within the same BC**.
- **If** teams share common layers, **they should only ever be technical /** cross cutting (e.g., security or infrastructure code)
  - **Caution:** Overindexing on the DRY principle *will* slow you down.

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Let's talk about boundaries (4)

© 2022, Amazon Web Services, Inc. or its affiliates.

Take boundaries seriously

As the saying goes: Fences make good neighbors

- Teams should **not share infrastructure**
- Infrastructure should always be **maintained as code**
- Caution: **Deployment pipelines** are also infrastructure!

Take boundaries seriously

- Shared infrastructure has implications:
  - Increased likelihood of **side-effects**
  - Increased **blast radius**
  - Increased need for **negotiation**
- Caution: Mind the **size** and **readability** of IaC files

© 2022, Amazon Web Services, Inc. or its affiliates.

Take boundaries seriously

- Consequently: **Align infrastructure with bounded contexts**
  - Use "**Infrastructure as actual code** (IaaC)" *(Gregor Hohpe)*
  - **Named variables / outputs** increase readability
  - Avoid "deployment repositories"
  - Include IAM roles/privileges to safeguard context boundaries

© 2022, Amazon Web Services, Inc. or its affiliates.

Take boundaries seriously

But… What about Kubernetes?

- Align helm charts and deployment descriptors with BC
  - Use IaaC to execute deployments and integrate with other components
- Prefer **self-service** over service-team-provisioned clusters
- Use cluster per team/BC to avoid **single point of failure** and minimize blast radius
- Consider **serverless container** offerings

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Take boundaries seriously

- Shared deployment pipelines are a **dependency**!
  - It is okay to use **common libraries**
    - Extract, don’t build from scratch!
    - Don’t enforce as standard
  - **Self-service** is a key element of success
    - Avoid gate-keeping and non-essential approvals at all costs

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Let’s talk about boundaries (5)

© 2022, Amazon Web Services, Inc. or its affiliates.

Data also needs context

Data can be classified into 3 categories, by usage type:

- **Operational** data
- **Analytical** data
- **Operationalized analytical** data

Different usage types require different architectural choices.

Data also needs context

- **Operational data** is the data generated / used by business applications
- Avoid central data stores, embrace eventual consistency and duplication
- Choose your persistence mechanism per context, by form and access type:
  - **Structured / curated** data used for **complex queries**: RDBS
  - **Unstructured** data **queried by ID** or as a **collection**: KVS / DocDB
  - **High volume / text search**: Indexed search engine
  - **Machine signals** / continuous data flow: Time series DB
  - Time projections / drill-down: Event store

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Data also needs context

- **Operational data** needs timely performance:
  - Be mindful of compound latency
  - Use caches if necessary, but be aware of additional complexity
  - Embrace CQRS
    - Can be implemented via Domain Events, technical events, polling…
      - Trade-off between convenience / maintenance cost / readability
    - Rule of thumb: **Limit write** functions to a single instance, but **scale read** functions generously

© 2022, Amazon Web Services, Inc. or its affiliates.

Data also needs context

- **Analytical data** is a **projection** (read model) of the operational data
- Used for business insights by:
  - **Business analysts** (to control/oversee, Data Warehouse)
  - **Data scientists** (to explore / discover, Data Lake)
  - Both need data from everywhere
    - Datasets must be **accessible**
    - Access must be **governed** (data protection)
    - Content must be **documented** and **searchable**

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Data also needs context

- **Remember: Platforms** are a dependency!
- Consequently: Data access should be **decentralized**
  - Dataset **scope** should align with BC
  - Curation of data is **team responsibility**
  - Expose datasets to externals via **self-service**
    - Requires metadata and access management
  - This is called **Data Mesh** *(Zhamak Dehghani)*

© 2022, Amazon Web Services, Inc. or its affiliates.

Data also needs context

- **Operationalized analytical data** is needed for AI/ML based decision making
- Data scientists discover ways to **predict outcomes** or **recognize entities** from patterns
  - Models are **trained** from **historical data**
  - Trained models are **operationalized** as Domain Services for inference on incoming operational data, become part of business applications
  - At this point, they become a **BC team responsibility**
  - Models incur **drift** and have to be re-trained

© 2022, Amazon Web Services, Inc. or its affiliates.

Data also needs context

- Manual model training is **slow** and **error-prone**
  - Also, it creates a **dependency** on the data science team!
- **Automation** (MLOps) is a key enabler
- **Caution:** ML is often **biased and unreliable** *(Abeba Birhane, @abebab)*
  - This is even more true for automated workflows

© 2022, Amazon Web Services, Inc. or its affiliates.

Let’s talk about boundaries (6)

© 2022, Amazon Web Services, Inc. or its affiliates.

Low Code / No Code – magic?

Low Code / No Code services promise **rapid results** and “focus on business”

- **Visual** / haptic interfaces
- Often bring their own **GUI / logic / persistence / deployment / hosting**
- Caution: Quick results often **sacrifice long-term sustainability**

Low Code / No Code – magic?

Integrated Low Code application platforms target non-programmers

- They usually offer a **one–stop-shop** solution from concept to release
- **Bias** towards CRUD type applications
- Pain points:
  - Versioning, refactoring, team integration
  - Debugging and observability
  - Data maintenance and access control
  - **Rule of thumb:** The more visual, the less extensible.

aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Step Functions

Step Functions are meant to be integrated with serverless applications

- “State machine as a service”, event-based
  - **Caution:** These are usually *not* equivalent to domain events
- Typical workflow patterns, e.g. pipeline, scatter/gather, human in the loop
- Integrates Lambda with many other services
- Visual interface / AWS Console

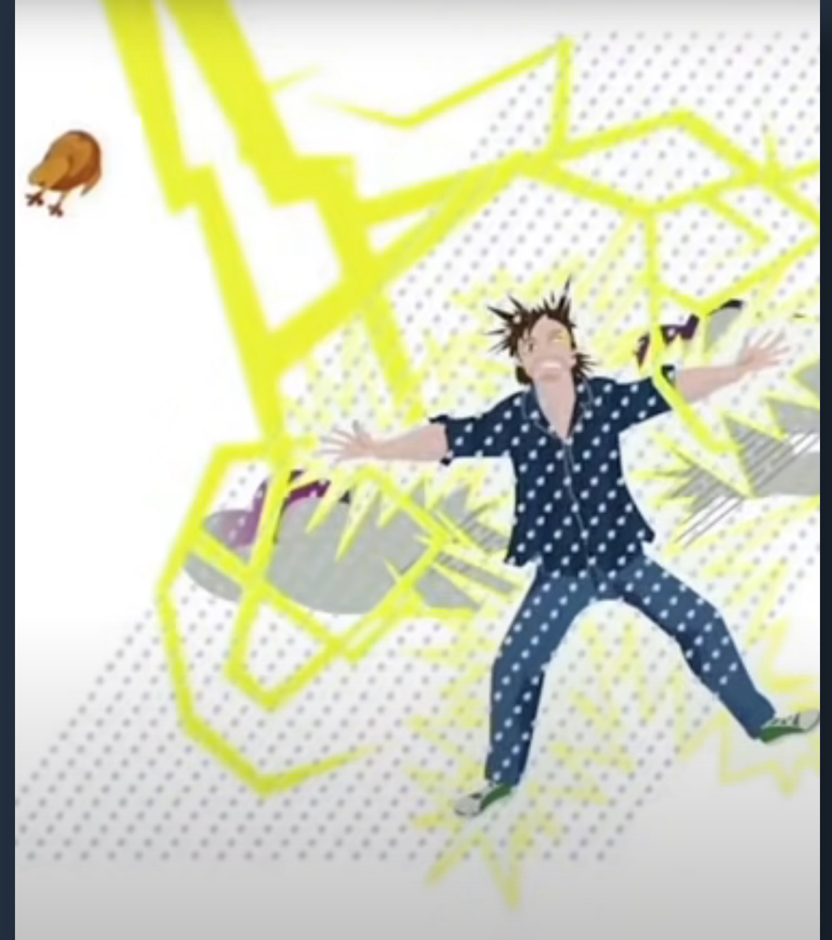
aws

© 2022, Amazon Web Services, Inc. or its affiliates.

Step Functions

- Business logic not explicit/separated
- Business rules in visual form / mixed with infrastructure code
- ”Magic” usually **hides complexity, but doesn’t make it disappear.**

My recommendation: Here be dragons. At least make sure you align with the bounded context.



aws
© 2022, Amazon Web Services, Inc. or its affiliates.

DDD in Cloud Native Environments – TL;DR

The fundamental principles of DDD still apply:

- The key enabler for scalable, extensible and evolvable systems is **knowing where to set good boundaries**.
    - To find them, pay attention to **language and context**.
- Keep domain logic **encapsulated and well-tested**.
- **Avoid dependencies across teams**, unless absolutely essential.
    - Do extract platforms and standards, but make them **optional**.
- Caution: **Typing is not the bottleneck!**

  (But Low Code / No Code can be useful when applied thoughtfully)

This is *still* difficult. To succeed, be prepared to iterate, inspect, and **learn**.

Thank you!

Tobias Goeschel

Sr Solutions Architect, FSI

Amazon Web Services

tgo@amazon.de

@w3ltraumpirat

© 2022, Amazon Web Services, Inc. or its affiliates.