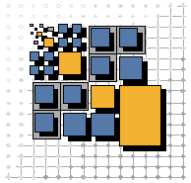


Evaluating Cloud-native Deployment Options with a Focus on Reliability Aspects

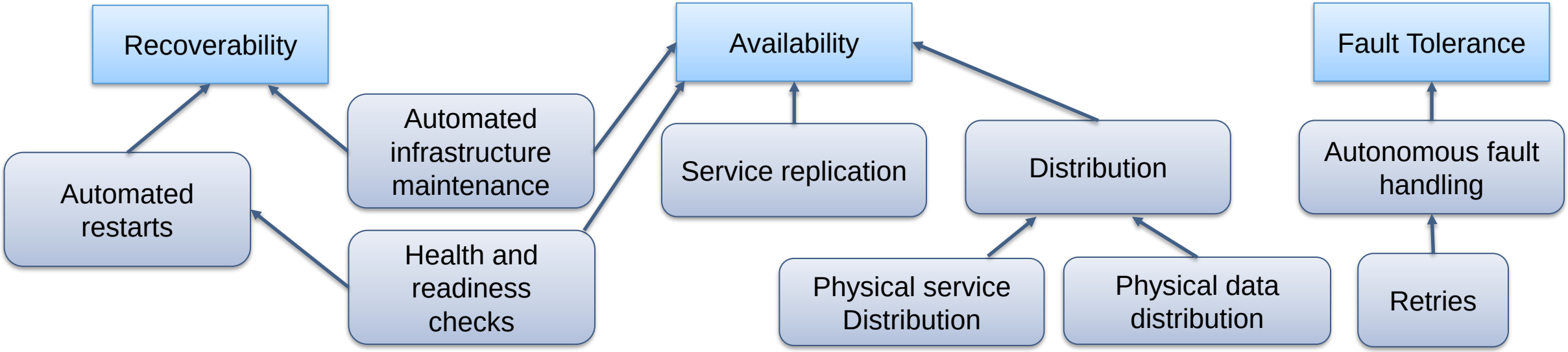
Franka Knoch, Robin Lichtenthäler, Guido Wirtz

01.07.2024

Distributed Systems Group
University of Bamberg

Motivation A Quality Model for cloud-native application architectures

Excerpt from the quality model (https://clounaq.de/quality-model):



Quality Aspect — *An abstract desirable observable behaviour of a system*

Product Factor — *An identifiable characteristic of software system*

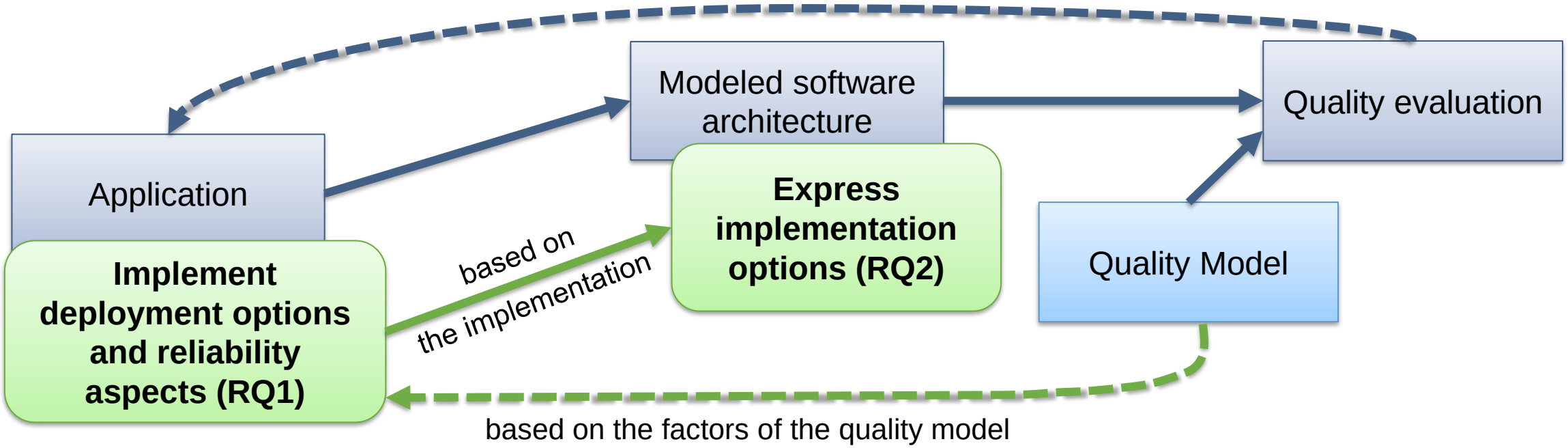
- Quality model formulated based on literature
- Intended as a basis to evaluate software architectures

Lichtenthäler, Robin / Wirtz, Guido: Formulating a quality model for cloud-native software architectures: conceptual and methodological considerations, 2024, *Cluster Computing,* 10.1007/s10586-024-04343-4

Approach Overview

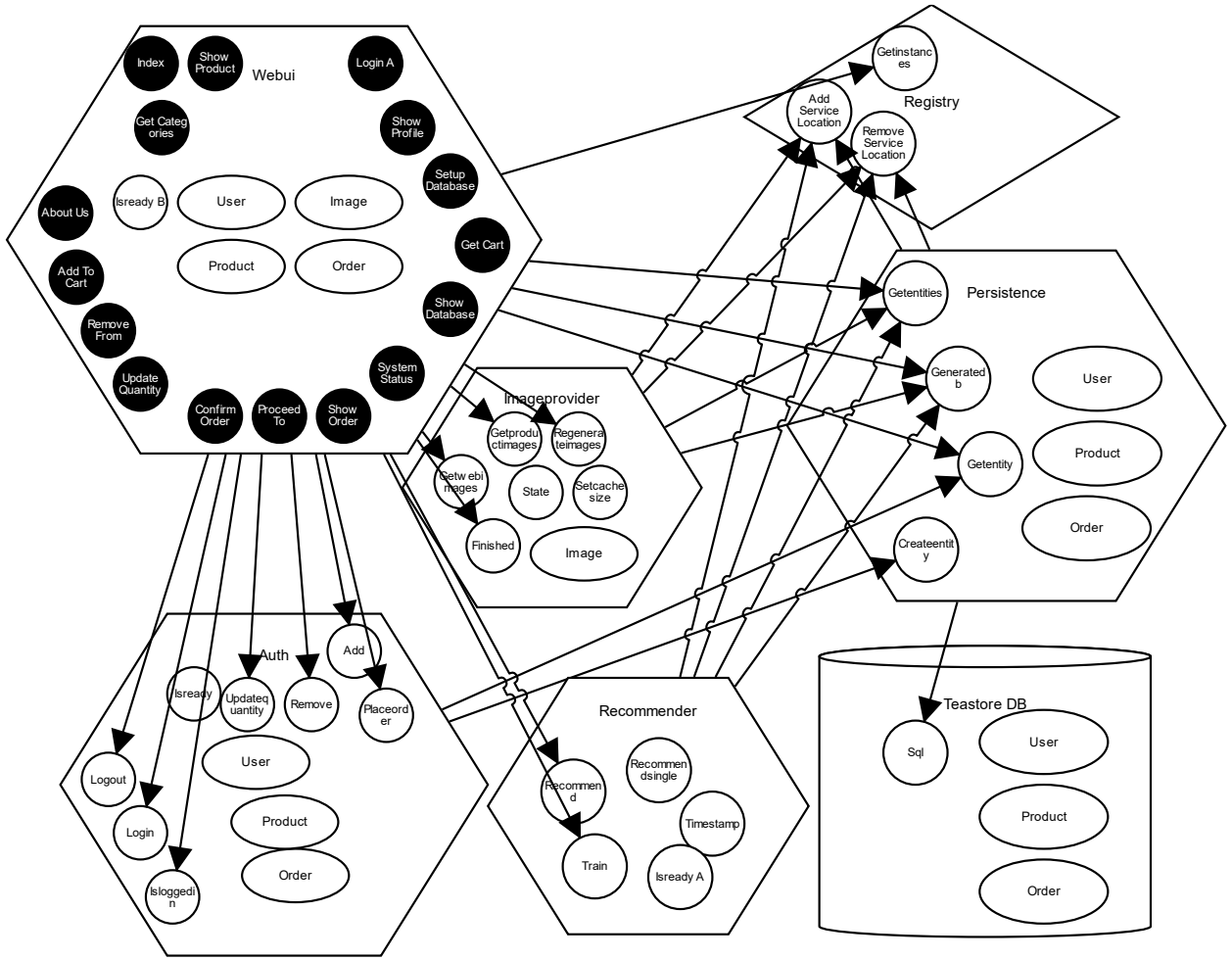
Overall goal: Evaluate software architectures of cloud-native applications according to quality aspects

In this work: **Extend the approach to express cloud-native characteristics with a focus on reliability**



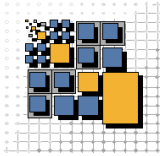
Approach Use Case Application: TeaStore

- TeaStore (https://github.com/DescartesResearch/TeaStore)
- Microservices reference application developed to enable benchmarking experiments
- Established and popular in research
- Features
  - Services implemented in Java
  - Communication via HTTP Calls
  - Relational database as a data store
  - Pre-configured Docker images and Kubernetes Deployment descriptions



von Kistowski, Joakim / Eismann, Simon / Schmitt, Norbert / Bauer, Andre / Grohmann, Johannes / Kounev, Samuel: TeaStore: A Micro-Service Reference Application for Benchmarking, Modeling and Resource Management Research, 2018, *26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE, p. 223-236, 10.1109/mascots.2018.00030

Results Implementation options for reliability aspects



Quality Aspect	Product Factor	Implementation option
Availability	Guarded Ingress	API Gateway | Ingress Controller | AWS WAF
	Service Distribution	Node Selectors | (Anti-) Affinity Rules | Pod topology spread constraints
	Data Distribution	StatefulSet | ManagedDatabase
	Built-In Autoscaling	EC2 Autoscaling Groups | K8s Cluster Autoscaler | Karpenter | Vertical Pod Autoscaler | Horizontal Pod Autoscaler
	Enforcement of Appropriate Resource Boundaries	Workload Annotation | Monitoring Services | Vertical Pod Autoscaler
	Seamless Upgrades	Rolling Upgrades | Blue-Green Strategy
	Health and Readiness Checks	Liveness and Readiness Probe | Container Health Checks
Recoverability	Automated Infrastructure Maintenance	Worker Node Versioning | Cluster Versioning
	Use Infrastructure as Code	AWS CloudFormation | Terraform

Results Exemplary implementation option

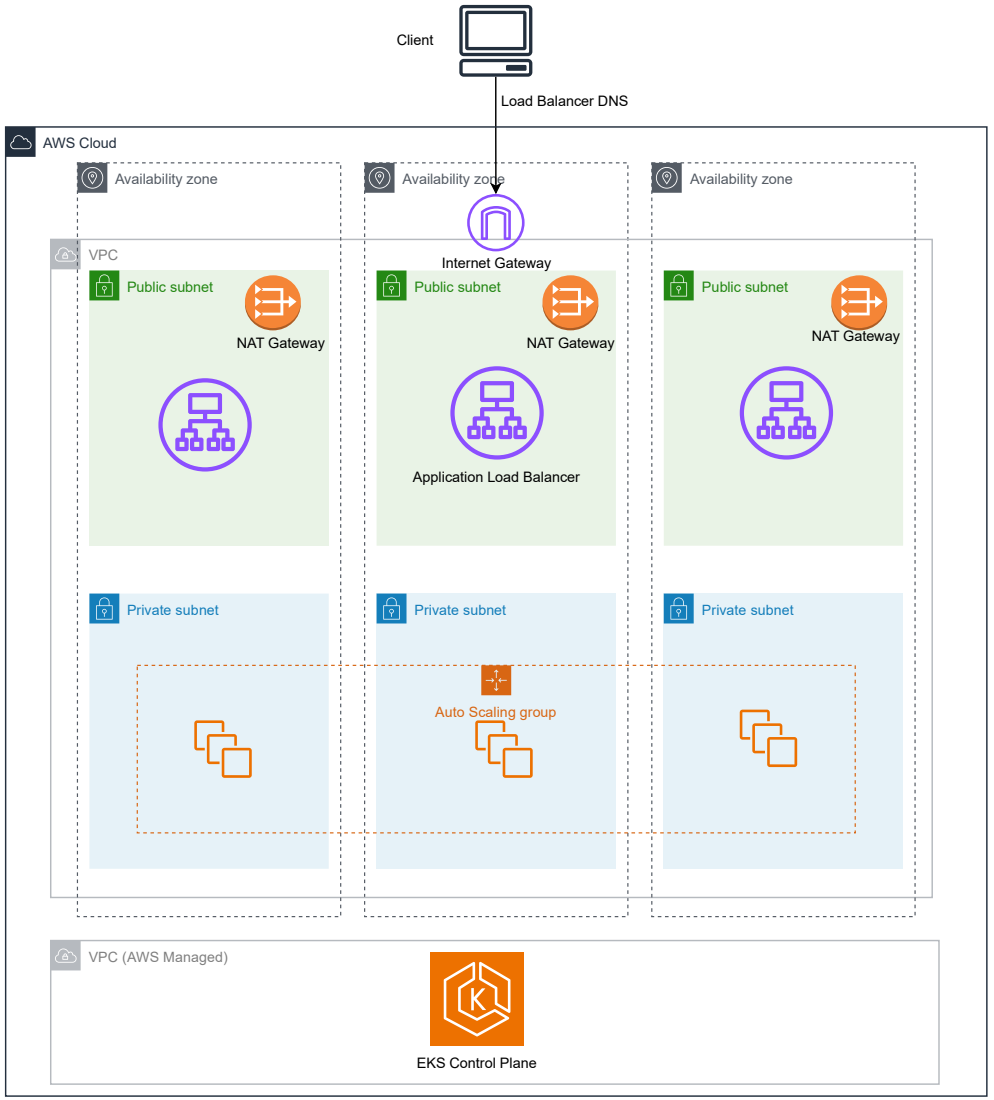
Physical Data Distribution via **Managed Database**

- TeaStore deployed on AWS Elastic Kubernetes Service (EKS)
- TeastoreDB provided as an AWS Relational Database Service (RDS) Instance and integrated into the Cluster via AWS Controllers for Kubernetes (ACK)

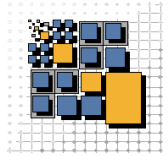
```
apiVersion: rds.services.k8s.aws/v1alpha1
kind: DBInstance
```

- Distribution through Multi-Availability-Zones flag and additional Read replicas

```
spec:
  multiAZ: true
---------------------
aws rds create-db-instance-read-replica \
    --availability-zone us-east-2a
```



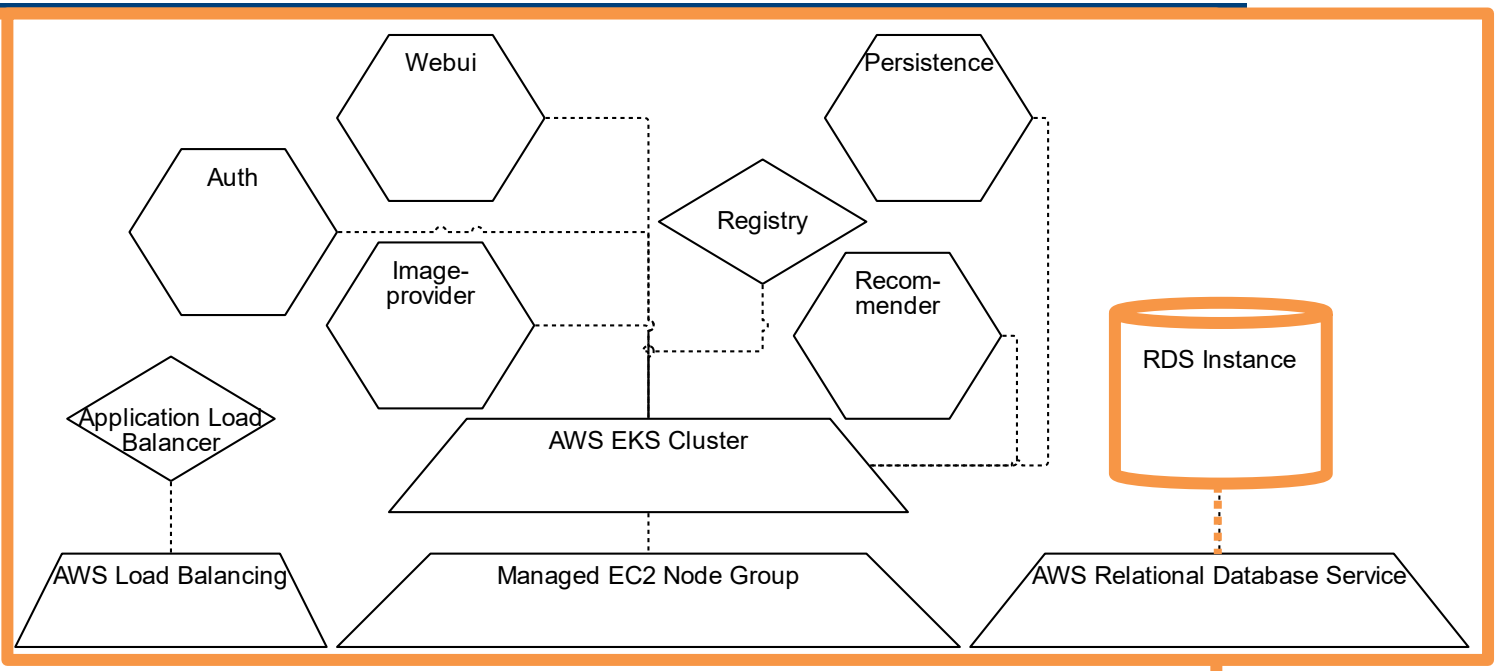
Results Representing cloud-native characteristics in a model



Entity	Example	Deployment extensions		Reliability extensions	
Component	-	*artifact* assigned_networks			
Service	WebUI			load_shedding *proxied_by*	
Backing Service	Registry				
Storage Backing Service	TeaStore DB				
Endpoint	Place order			rate_limiting readiness_check	health_check idempotence
External Endpoint	Show products				
Link	WebUI → Place Order			timeout circuit_breaker	retries
Infrastructure	AWS EKS	kind environment_access maintenance	provisioning supported_artifacts assigned_networks	availability_zone region deployed_entities_scaling	self_scaling supported_update_strategies enforced_resource_bounds
Deployment Mapping	WebUI → AWS EKS	deployment deployment_unit	assigned_account	update_strategy automated_restart_policy	resource_requirements replicas
Data Aggregate	Order			usage_relation	

Results Exemplary representation in the model

```
rds_instance:
  type: cna-modeling.entities.StorageBackingService
  properties:
    managed: true
    software_type: proprietary
    stateless: false
    load_shedding: false
    assigned_networks:
      - teastore-private-1
      - teastore-private-2
      - teastore-private-3
  artifacts:
    rds_instance:
      type: AWS.RDS.Instance
  requirements:
    - host:
        node: aws_relational_database_service
        relationship: aws_relational_database_service_hosts_rds_instance
aws_relational_database_service_hosts_rds_instance:
  type: cna-modeling.entities.HostedOn.DeploymentMapping
  properties:
    deployment: automated-declarative
    deployment_unit: RDS Instance
    replicas: 3
    update_strategy: blue-green
    automated_restart_policy: onProcessFailure
    assigned_account: default-account
    resource_requirements: unstated
```




```
aws_relational_database_service:
  type: cna-modeling.entities.Infrastructure
  properties:
    kind: cloud-service
    environment_access: none
    maintenance: transparent
    provisioning: transparent
    supported_artifacts:
      - AWS.RDS.Instance
    availability_zone: us-east-1a,us-east-1b,us-east-1c
```

Limitations & Future Work

- One cloud provider (AWS) and one application considered
  → Extend to other providers and applications
- Focus on reliability
  → Integration of quality aspects in addition to reliability
- Usage of TOSCA only for modeling, not for deployment.
  Ideally both should be possible, but there are challenges
  → Happy to discuss with you at the poster session

- Further development and validation of the overall approach

Conclusion

- Various implementation possibilities for cloud-native characteristics
  - ⚠ Often specific to certain technologies or cloud provider offerings
  - ⚠ Interdependencies between decisions for different options
  - → Abstraction and comparability is difficult
  - → Contribution of a set of practical implementations as a data basis
- For the chosen modeling approach a level of abstraction needs to be chosen (inherent challenge of modeling)
  - If too abstract, differences between implementation options diminish
  - If too detailed, modeling becomes an effort
  - → Focus on major differentiating aspects

More detailed information online

https://github.com/frankakn/cloud-native-deployment

Implementations and models with more detailed descriptions

https://clounaq.de

Tooling aimed at supporting the overall approach (in development)

Thank you for your attention!