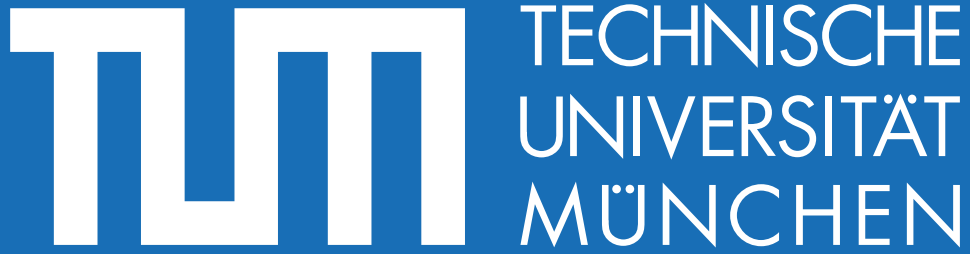


Software Engineering in der industriellen Praxis (SEIP)

Dr. Ralf S. Engelschall

ARCHITECTURE FUNDAMENTALS

Software Deployment

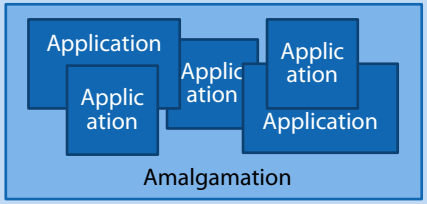
TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 13.1

AMA Bare Amalgamation

Manually deploy all applications into a single, shared, and unmanaged filesystem location. Dependencies are resolved manually. Examples: Windows Fonts, Unix 1990th /usr/local.

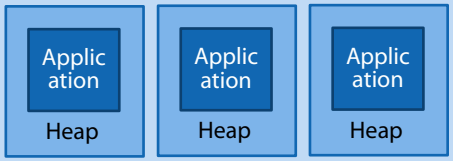
Pro: simple deployment
Con: incompatibilities, hard uninstallation



UHP Unmanaged Heap

Manually deploy all applications into multiple, distinct, and unmanaged filesystem locations. Dependencies are resolved manually. Examples: macOS *.app, OpenPKG LSYNC.

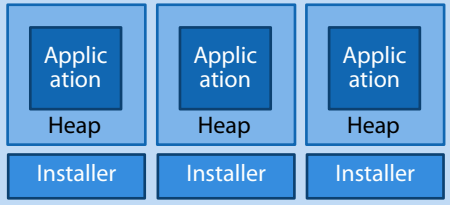
Pro: simple deployment, easy uninstallation
Con: no repair mechanism



MHP Managed Heap

Let individual installers deploy applications into multiple, distinct, and managed filesystem locations. Dependencies are manually resolved or bundled. Examples: macOS *.pkg, Windows MSI, InnoSetup.

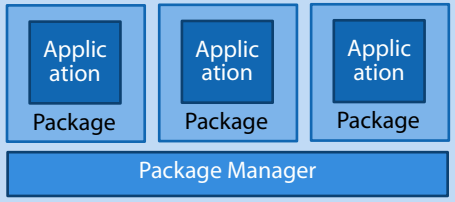
Pro: easy uninstallation, repairable
Con: requires installer, diversity, no dep.



PKG Managed Package

Let a central package manager deploy all applications into a single, shared, and managed filesystem location. Dependencies are automatically resolved. Examples: APT, RPM, FreeBSD pkg, MacPorts, Gradle, NPM.

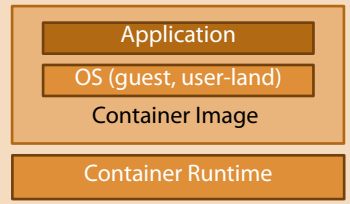
Pro: easy uninstall., repairable, dependencies
Con: P.M. pre-installation, P.M. single instance



CON Container Image

Bundle an application with its stripped-down OS dependencies and run-time environment into a container image. Examples: Docker/ ContainerD, Kubernetes/CRI-O, Windows Portable Apps.

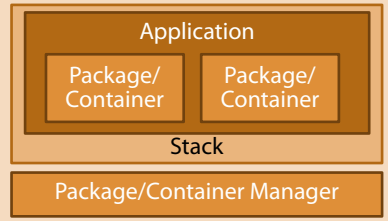
Pro: independent, simple deployment
Con: fewer variations, no dependencies



STK Package/Container Stack

Establish an application out of multiple Managed Packages. Examples: OpenPKG Stack, Docker Compose, Kubernetes/ Kompose, Kubernetes/Helm.

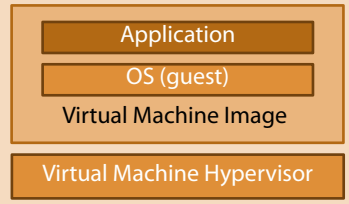
Pro: independent, flexible
Con: overhead



VMI Virtual Machine Image

Bundle an application with its full OS dependencies and run-time environment into a virtual machine image and deploy and execute this on a hypervisor. Examples: VirtualBox, VMWare, HyperV, Parallels, QEMU.

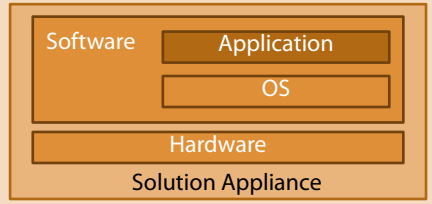
Pro: all-in-one, independent
Con: overhead, sealed, inflexible



APP Solution Appliance

Bundle an application with its full OS dependencies, run-time environment and underlying hardware. Examples: AVM Fritz! Box, SAP HANA.

Pro: all-in-one, independent
Con: expensive, sealed, inflexible



ARCHITECTURE FUNDAMENTALS

Cloud Computing Resources

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 13.2

Cloud Characteristics — How?

logically / chronologically related

- ON-DEMAND
- SELF-SERVICE
- AUTOMATABLE
- NETWORK ACCESS

logically / chronologically related

- MULTI-TENANCY
- MEASURED SERVICE
- ELASTICITY
- PAY-PER-USE

Cloud Scope — Who?

outside-inside relationship

- Public Cloud
- Community Cloud
- Private Cloud

Cloud Location — Where?

outside-inside relationship

- Provider-Hosted
- On-Premises

Cloud Approaches — What?

technical layers

- Service
- Application
- Service Mesh
- Service Events
- Application Runtime
- Tools & Libraries
- Container Runtime
- Operating System
- Computer
- Network

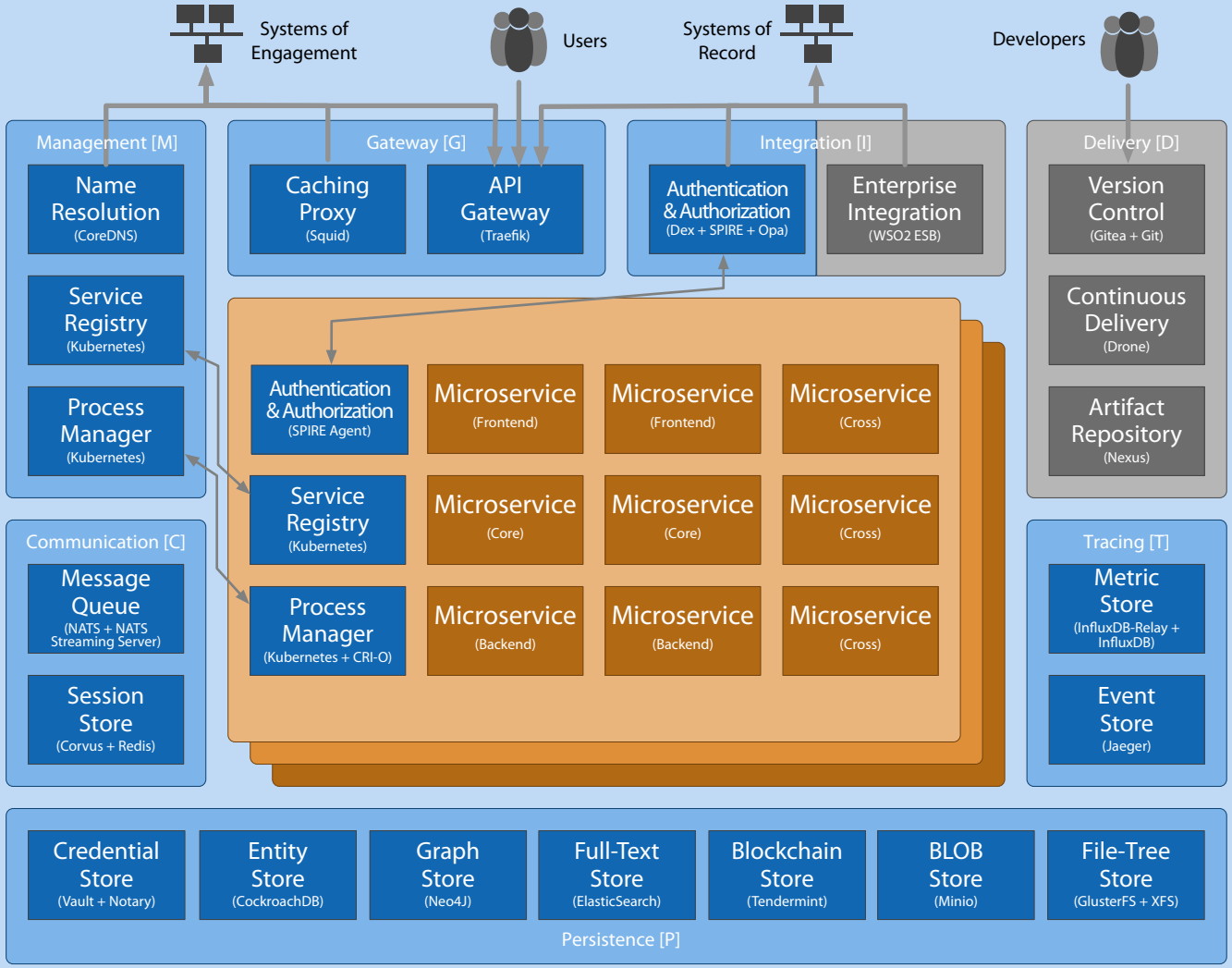
	SaaS	FaaS	PaaS	CaaS	IaaS
	e.g. Salesforce e.g. Sharepoint 365 (think: Web-UI-based Service)	e.g. AWS Lambda e.g. Azure Functions (think: Stateless Microservice)	e.g. AWS EKS e.g. Azure AKS (think: Kubernetes)	e.g. AWS ECS & Fargate e.g. Azure Container Apps (think: Docker)	e.g. AWS EC2 e.g. Azure Virtual Machines (think: Virtual Machine)
		aka "Serverless Computing"			
	Software as a Service	**Function** as a Service	**Platform** as a Service	**Container** as a Service	**Infrastructure** as a Service
Duration:	User Session	Function Call	Continuously		
Interfaces:	Proprietary			Standardized	
				(think: OCI)	(think: PC)

ARCHITECTURE FUNDAMENTALS

Cloud-Native Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

Reference Architecture Blueprint



Major Approach Idea:
With Cloud-Native Architecture one maximizes the leverage of PaaS-like, high-available, and scalable Cloud services at the level of Software- and Systems-Architecture for a whole set of applications.

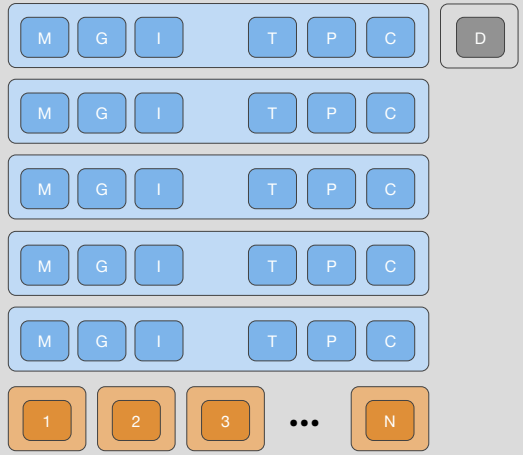
Major Design Criterias:
1. Targets DevOps approach.
2. Targets Continuous Delivery process.
3. Targets Microservice Architecture.
4. Targets Container Image deployment.
5. Targets Service Mesh communication.
6. Targets Server Cluster setup.
7. Provides High-Availability of Service Platform
8. Provides High-Availability of Application Microservices.
9. Provides Scalability of Application Microservices.

CNCF Cloud-Native Definition 1.0

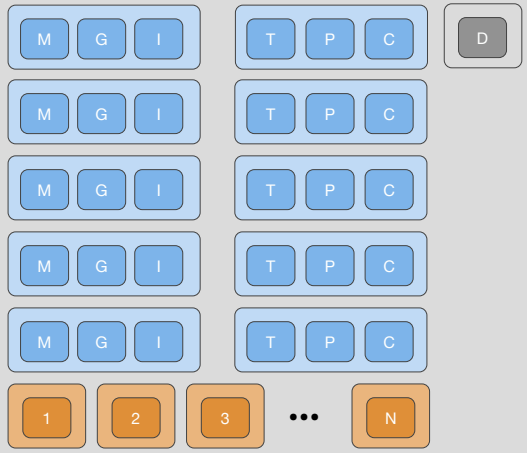
Cloud-native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach. These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

Practical Cluster Setups

Standard Cluster Setup (5+1+N Machines):



Partitioned Cluster Setup (5x2+1+N Machines):



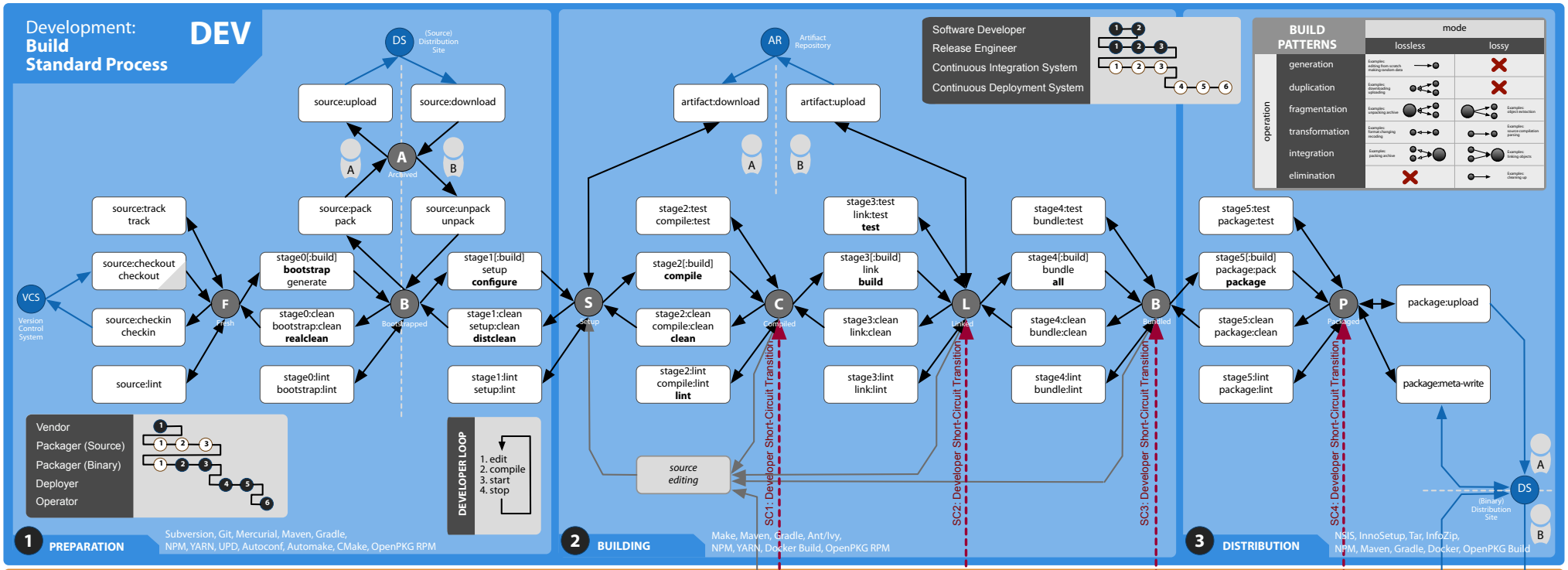
AF 13.3

ARCHITECTURE FUNDAMENTALS

Assembly Process Architecture

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 14.2

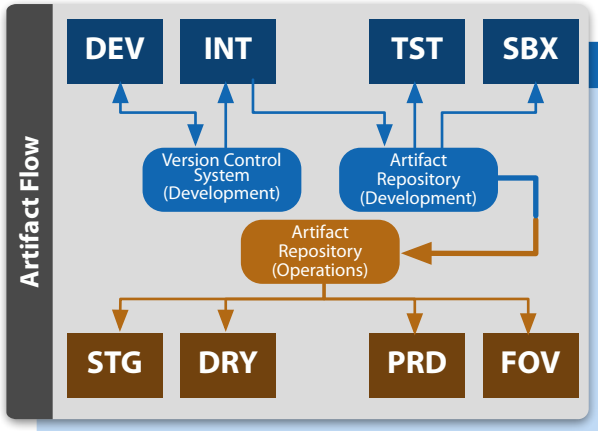


ARCHITECTURE FUNDAMENTALS

Environments & Quality Assurance

TECHNISCHE UNIVERSITÄT MÜNCHEN

AF 14.3



Development Environments

DEV Development

(Potentially destructive) Environment for the (separated) development of all artifacts of the system, typically located on each developer's respective system.

INT Integration

(Potentially destructive) Environment for integrating all artifacts of the system, usually located centrally to the developers and driven automatically.

SBX Sandbox

Destructive environment mirroring the Test environment for demonstration and experimentation purposes, including trainings and showcases.

TST Test

Environment for testing the system as a whole in a production-resembling context.

Operations Environments

DRY Dry-Run

Destructive environment mirroring the Staging environment for experimentation purposes, including operating system and major dependency upgrades.

STG Staging

Environment for staging the system as a whole in a production-equal context in order to deploy it to the Production environment subsequently.

Environment Mirroring

FOV Failover

Destructive environment mirroring the Production environment for failover situation, including disaster recovery situations.

PRD Production

Environment for running the system as a whole in production.

Replicated Environments

Original Environments

Testing Targets:

Quality Promise, Bugfix Effort

Testing Approaches

UT Unit Testing

Whitebox testing the inner details of individual units (components) against their functional specification.
Specialization: Regression Testing

IT Integration Testing

Whitebox testing the outer interplay of individual units (components) against the technical design of the solution.
Specialization: Smoke Testing

ST System Testing

Blackbox testing the system as a whole against the functional/domain and non-functional/quality requirements of the solution.

UAT (User) Acceptance Testing

Blackbox testing and formal approval of the system as a whole against the end-to-end functionality and user experience requirements.

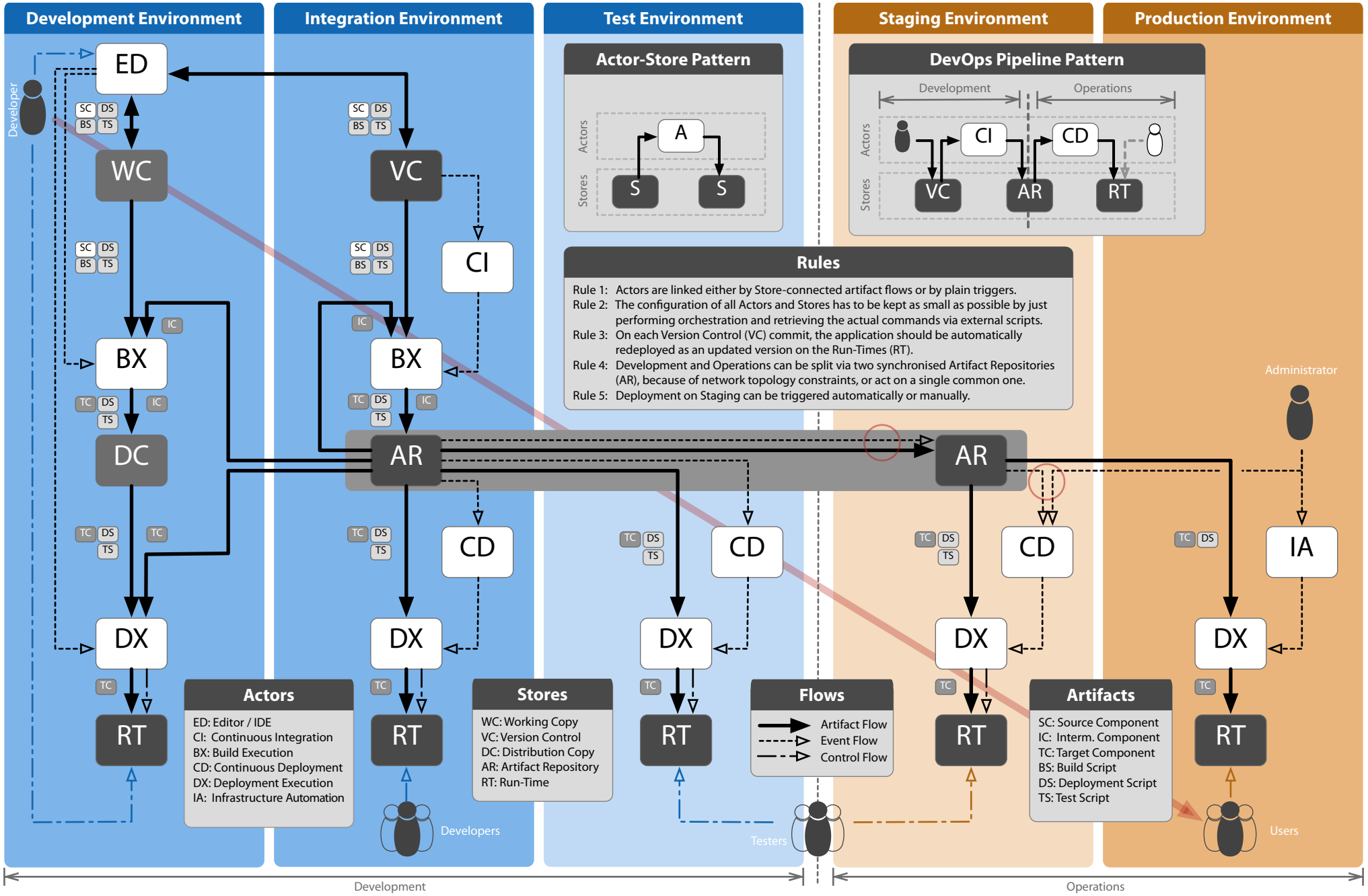
OT Operation Testing

Blackbox testing the system as a whole against the availability and proper operation of the intended services.
Specialization: Service Monitoring

ARCHITECTURE FUNDAMENTALS

DevOps Toolchain

TECHNISCHE UNIVERSITÄT MÜNCHEN



ARCHITECTURE FUNDAMENTALS

Software Release Management

TECHNISCHE UNIVERSITÄT MÜNCHEN

Evolution Stages (ES) — What?

WF Wireframe
Distraction-free low-fidelity illustration of the solution and its base features, displaying its pure structure and core functionality only.

WS Walking Skeleton
Realization of all technical, fundamental aspects of the solution, ensuring the domain specific aspects can be realized later on top of it without problems.

PT Prototype
High-fidelity mostly interactive sample, mockup, model or simulation of the solution and its base features, showcasing its structure and functionality.

MVP Minimum-Viable Product
Early version of solution with just enough functionality to enable full turn of Build-Measure-Learn loop with minimal amount of effort and time.

PoC Proof of Concept
Pure realization of most-risky aspects of the solution, proofing their feasibilities. Might still be based on a different technology than WS, MVP and FP.

FP Full Product
Final version of the solution with all intended functionality and targeting the mainstream market.

arbitrary technology | target technology

Version Number (VN) — When?

M Major Version
Major Version of solution. Usually bumped on major technical or domain-specific changes only. A bump resets the Minor Version and the Revision, too.

N Minor Version
Minor Version of the solution within the Major Version. Usually bumped on new features. A bump resets the Revision, too.

K Revision
The Revision of the Maturity Level within Major and Minor Version. Bumped for every A/B/RC/R/RU Maturity Level.

Maturity Levels (ML) — When?

A Alpha
Early version of the solution with incomplete and unstable functionalities to get feedback on product. Usually tagged as "M.NaK" (K > 0).

B Beta
Early version of the solution with complete but still unstable functionalities to stabilise product. Usually tagged as "M.NbK" (K > 0).

RC Release Candidate
Mature version of solution with complete and stable functionalities to catch last-minute problems. Usually tagged as "M.NrcK" (K > 0) around RTM.

R Release
Final version of the solution with complete and stable functionalities, available for production use. Usually tagged as "M.N.0".

RU Release Update
Updated version of the solution with complete and stable functionalities, available for production use. Usually tagged as "M.N.K" (K > 0).

Points-In-Time (PiT) — When?

DEV Development
Arbitrary permanent points-in-time during development. This is the default tag for the source code. Intended for no availability releases.

SNP Snapshot
Distinct temporary point-in-time for a release of the current version without a version increase. Intended for limited availability releases.

REL Release
Distinct temporary point-in-time for a release of the current version with a version increase. Intended for early and general availability releases.

Product Editions (PE) — Which?

CE Community Edition
Edition of the solution for the Open Source Community. Contains just the base functionality and has limited volunteering support.

STD Standard Edition
Edition of the solution with just the standard functionality and regular support.

EE Enterprise Edition
Edition of the solution for the Enterprise market. Contains the base and additional functionality and has full commercial support.

PRO Professional Edition
Edition of the solution with both the standard and extra functionalities and extended support.

Availability Scopes (AS) — Who?

XA No Availability
No public availability of solution at all. The scope for all Development and sometimes Snapshot point-in-times.

EA Early Availability
Early public availability of solution for early market. Usually for Beta or Release Candidate levels or for Release and initial Release Update levels.

LA Limited Availability
Limited public availability of solution. Usually for releases after the End-of-Life-Announcement (EOLA) or for releases with specific customer features.

GA General Availability
Late public availability of solution for mainstream market. Usually for Release and sometimes just for Release Update levels.

Distribution Channels (DC) — Where?

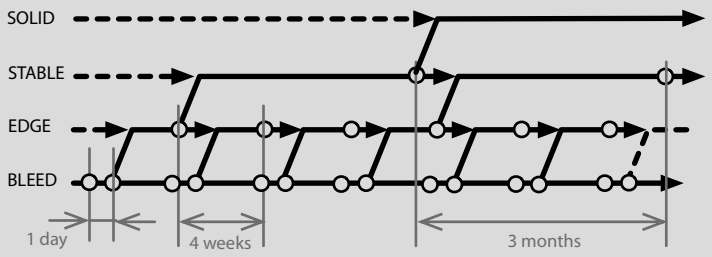
BLEED Bleed Channel
Distribution channel for all daily snapshots ("YYYY.MM.DD") with experimental features turned on. Intended for testing purposes only.

STABLE Stable Channel
Distribution channel for all quarterly releases ("YYYY.QN") with experimental features turned off. Intended for fast mainstream market and production use.

EDGE Edge Channel
Distribution channel for all monthly releases ("YYYY.MM") with experimental features turned on. Intended for early market or testing purposes.

SOLID Solid Channel
Distribution channel for all (half-)year releases ("YYYY[.N]") with experimental features turned off. Intended for slow mainstream market and production use.

Version Control (VC)

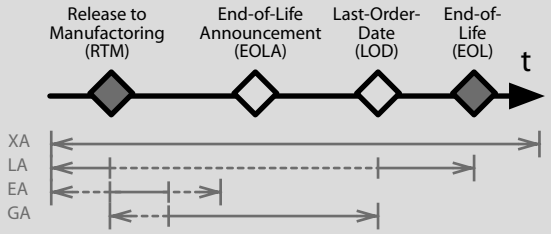


Version Scheme (VS)

***<Name>*[/*<ES>*]**
***<M>*.*<N>*<*ML*><*R*>**
[-*<PiT>*[-*<PE>*[-*<AS>*[-*<Date>*]]]]

Foo 1.2a3-DEV
Foo 1.2.3-REL-EE-GA
Foo/PT 1.2b3-SNP-CE-LA-2018.01.01

Product Life-Cycle (PLC)



AF 15.2