CREDIT SUISSE

JCP EC Meeting: May 14+15, 2013

Using Java in Credit Suisse

Introduction to CS Standard Platforms

Infrastructure Architecture & Strategy,
Susanne Cech Previtali (KIVO) and Peter Schnorf (KIVB)

May 15, 2013

Credit Suisse Group today – Key Facts (02/2013)

- Global bank headquartered in Zurich, serving clients in private banking, investment banking and asset management.
- Registered shares of Credit Suisse Group AG (CSGN) are listed in Switzerland (SIX) and as American Depositary Shares (CS) in New York (NYSE).
- Total number of employees: 47,400.
- The Group's long-term ratings are: Moody's A2, Standard & Poor's A, Fitch Ratings A.



Outline

1. Credit Suisse Platforms
2. Java Application Platform
3. Our Vision: JAP in the Cloud
4. A Customer's Perspective

Outline

Motivation Behind Platforms

Recurring themes in application development and operations

What infrastructure software (SW) components to use (and which versions)?

How to monitor correct behavior and find/fix incorrect behavior?

How to communicate with other application components?

Which hardware (HW) components and system setup to use to achieve the necessary performance, availability, stability, security, IT DR level, etc. at the right price?

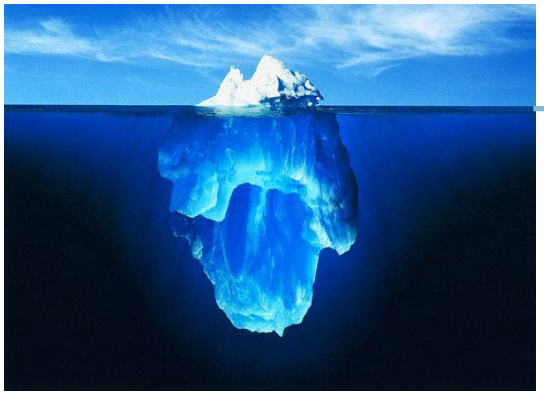
How to stay current in technology life cycle and avoid out-of-support legacy?

Which infrastructure SW component features to use?

How to manage changes to infrastructure SW components as well as to application components?

CS Platforms – Motivation and Definitions

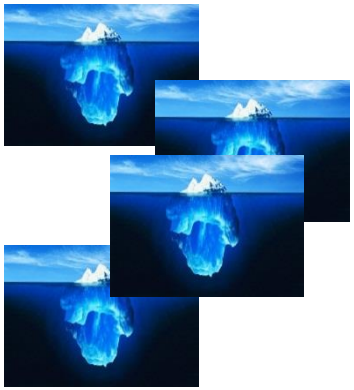
Infrastructure: Standardize, Build Once, Automate, Reuse



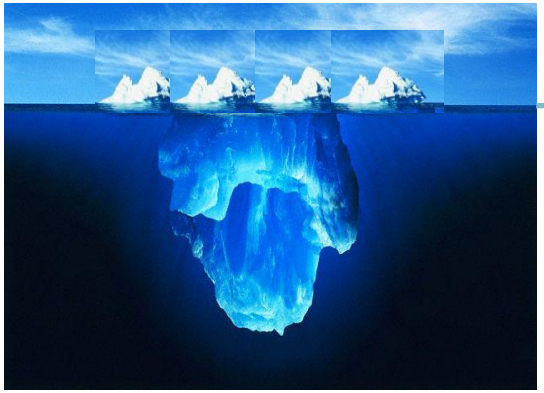
Application Specific Logic
GUI, Business Logic, DB Schemas, Configuration, etc.

Infrastructure Design/Configuration
HW, OS, Middleware, Network
System Management setup and processes
Operating manual
Development tools and processes
Security concept and processes
Integration concept and processes

CS Platform
Set of integrated technical components, processes, guidelines for the development and operation of applications



standardize
build once
automate
reuse

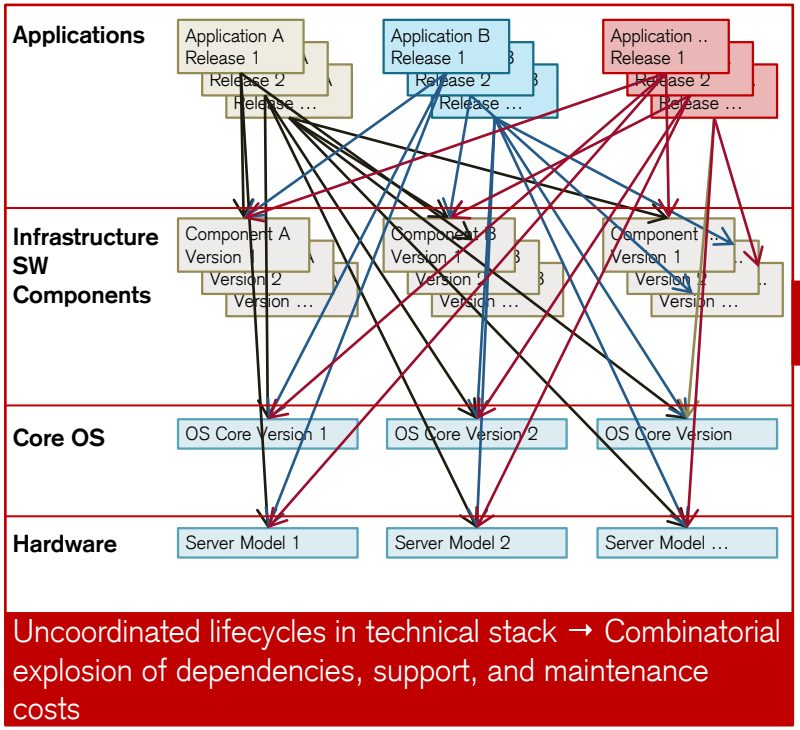


Application Platform (AP)
specialized for similar applications, built on hosting platforms

Hosting Platforms
provide generic services
→ computation CHP
→ persistence DHP

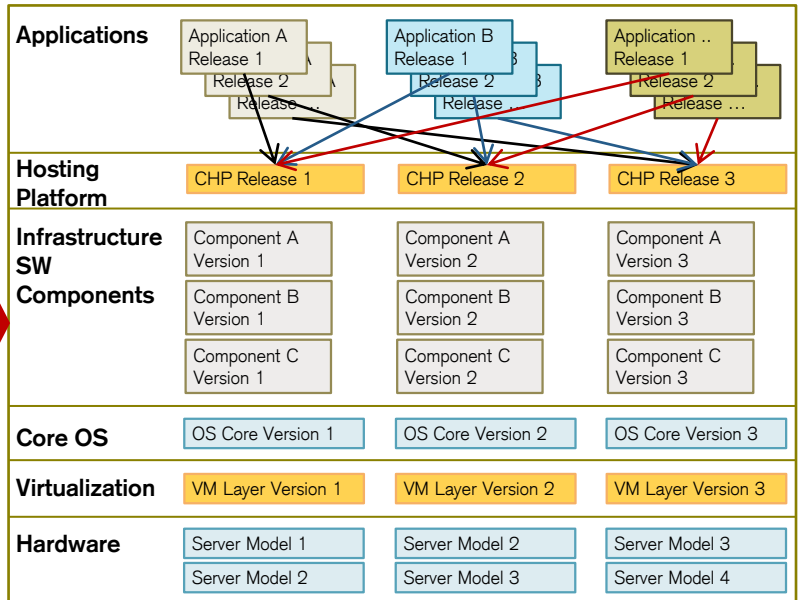
CS Platforms - Portfolio Management Drivers

Benefits of managing a platform lifecycle include increased stability and reduced cost

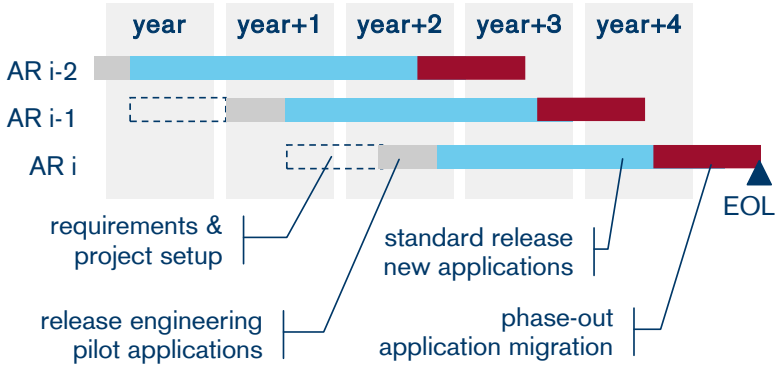


Uncoordinated lifecycles in technical stack → Combinatorial explosion of dependencies, support, and maintenance costs

Platform Release Mgmt.



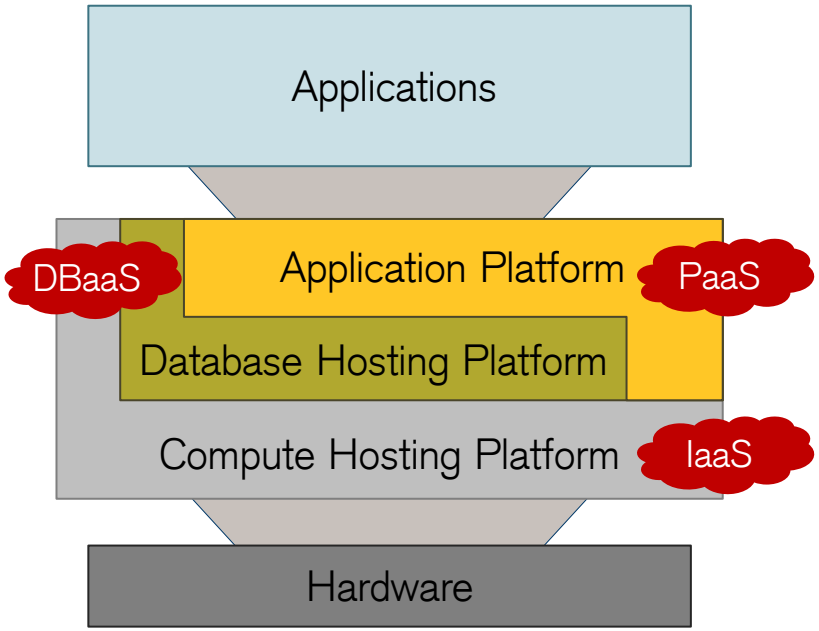
Managed stack → No dependency explosion, increased stability, capacity management, less maintenance costs, less support



Key Benefits of Platform Lifecycle Mgmt.

- Applications (and platforms) stay in technology lifecycle and mainstream (no "rotten" components)
- Technical upgrades due to lifecycle mgmt. of platform are combined with update on business functionality
- Constant decommissioning of out-dated platform releases identifies out-of-use applications
- New releases of platforms (with new features) have no impact on productive applications (no need to migrate; no stability impact due to changes)

CS Platforms Overview



- **Application Platforms (JAP, DAP, DWH)**
  - Specialized for application areas with similar needs
  - Support standardized application development and operations
  - Provide enhanced and additional services on top of generic Hosting Platforms
  - Platform costs are amortized by cost savings across large enough application portfolio using it
- **Database Hosting Platform (DHP)**
  - Offer DB service and backend system with decoupled life cycle
- **Compute Hosting Platform (CHP)**
  - Underlying infrastructure basis for directly deployed applications (typically 3rd party), Database Hosting Platforms as well Application Platforms
  - Abstraction of infrastructure services for independent lifecycles of infrastructures and applications

Reduce development risk by offering standardized, integrated, and tested infrastructure stack and related processes

CS Platform Benefits

Benefits of standardization and upfront investment in infrastructure stack/processes

Objectives

- Design, build and test standard infrastructure stack (including middleware) once
  - "Sharing the stack": Amortize over many applications
- Standardize interfaces between applications and infrastructure
- Design well-defined, highly automated processes using standardized ecosystems
- Strict release and whole platform lifecycle management (all components/processes at once)
- Global availability

Benefits

- Lower cost
  - Reduced development, maintenance, and support cost for applications and infrastructure
- Better quality
  - Increased infrastructure and application stability
- Lower risk
  - No end-of-life technologies and components in data center
  - Increased application security
  - Reduced development risk
- Enhanced capability
  - Shorter time-to-market
  - Global deployment of applications

Platform Contract: Adherence to Platform Release and Lifecycle Management

- New applications and application upgrades must use the latest major platform release available
- All applications using a platform release x must migrate to the latest platform release before the end-of-life date of release x is reached

→ ***Use the roadmaps to plan the releases of the applications***

Outline

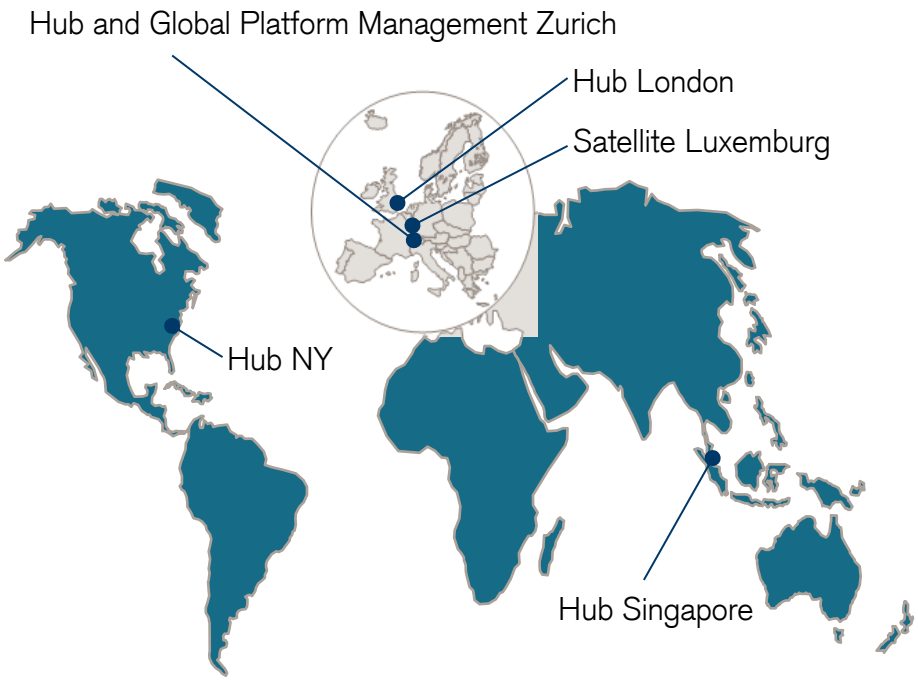
1 Credit Suisse Platforms

2 Java Application Platform

3 Our Vision: JAP in the Cloud

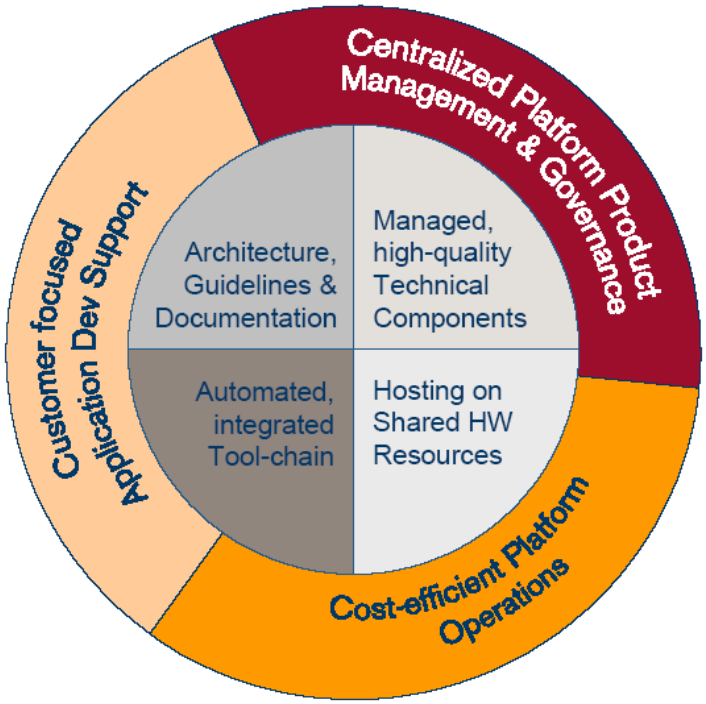
4 A Customer's Perspective

JAP Facts and Figures



- JAP is an Application Platform based on Java EE standards specialized for Web applications (both Intranet and Internet facing) and transactional systems
- JAP is providing services to more than 300 applications of all sizes, worldwide and across all business units
- Geographical presence
  - Platform centrally managed out of Zurich
  - 4 hubs providing consulting services as well as complete integration test, UAT and production environments
- Lifecycle & Availability
  - 3 major releases in parallel
  - AR 7 is the most recent release
  - AR 5 release in phase out

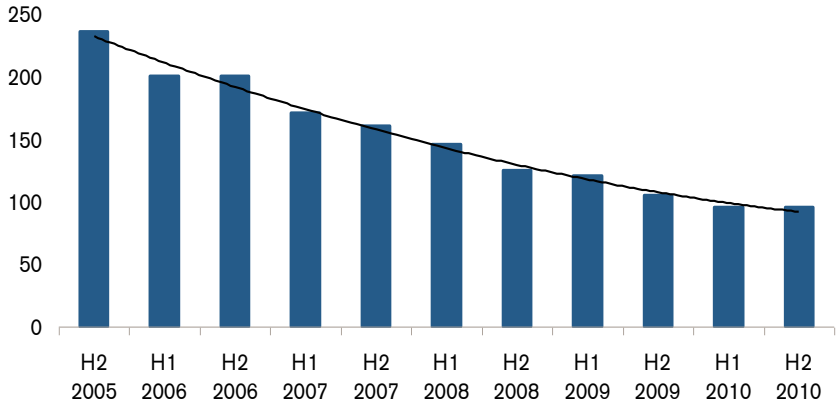
JAP Model

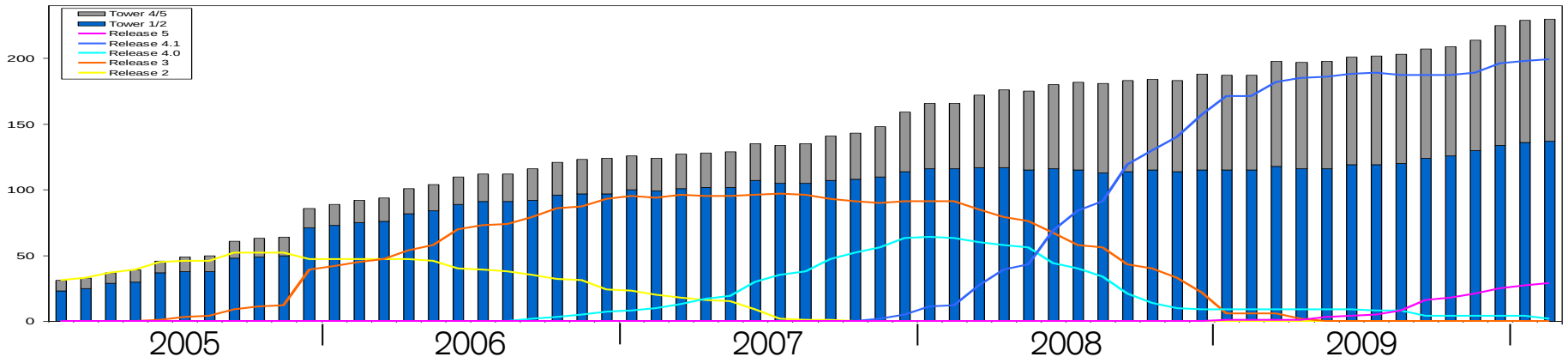


- Services provided by JAP
  - **Platform Product Mgmt. & Governance:** Drives product development and release & lifecycle
  - **Application Development Support:** Consultants guides projects through entire development processes, Java development support and trainings
  - **Platform Operations:** Cost-efficient standardized and/or automated processes according to a defined set of OLA's
- Key components to provide these services
  - **Technical Components:** Java EE based software stack pre-integrated with security, middleware, databases etc.
  - **Hosting:** Shared hardware resources according to production guidelines
  - **Tool-chain:** Automated processes from configuration management, centralized builds, package generation to deployment
  - **Architecture, Guidelines & Documentation:** security, IT-DR, HA designed and provided for all applications, transaction processing incl. patterns, scalability via LBs and scale-out

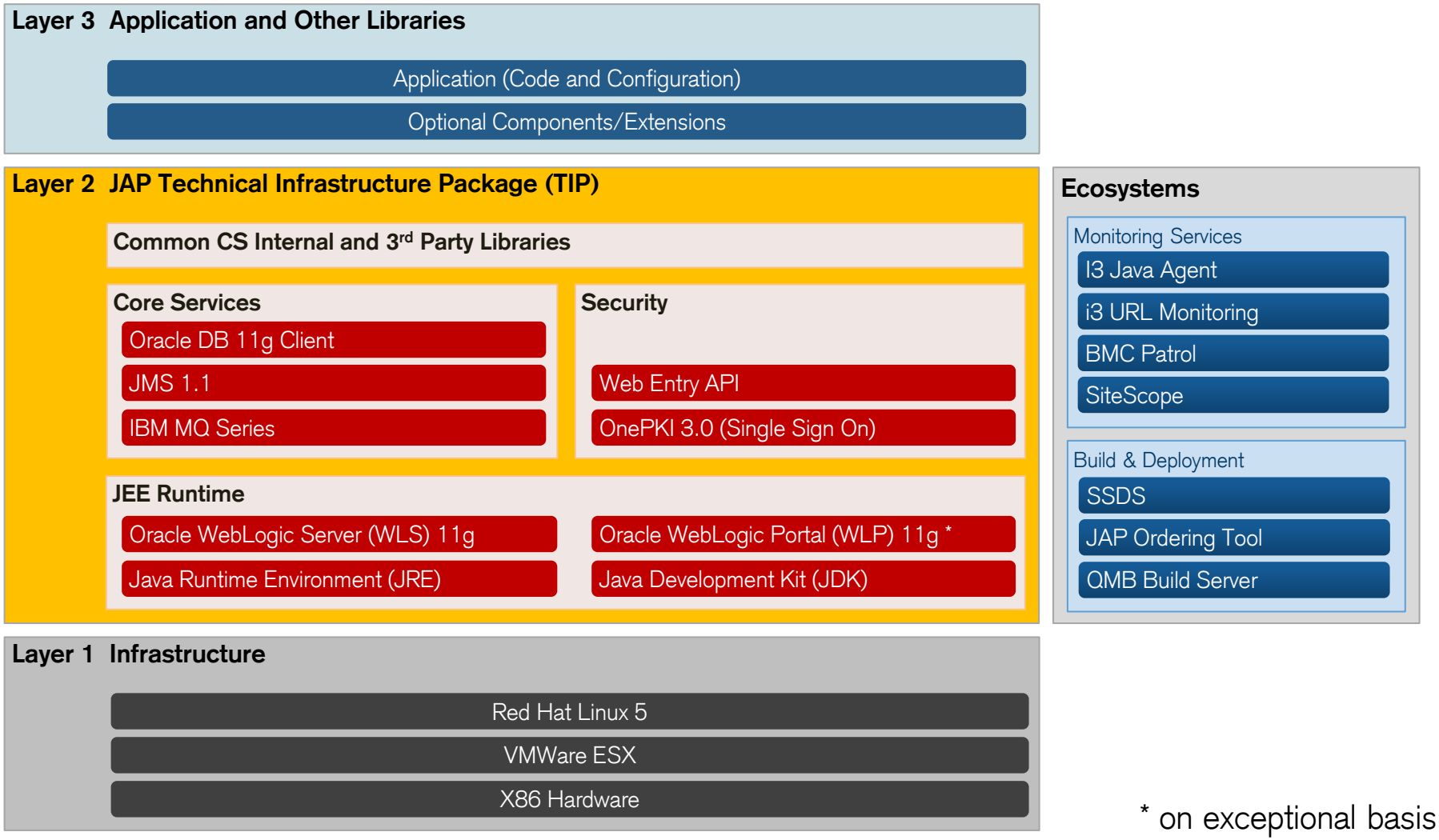
Platform Lifecycle in Action

- Applications benefit in terms of
  - Reduction of 30% on project costs (CTB budget)
  - Reduction of 35% on operating costs (RTB budget)
- JAP Hub Zurich realized
  - Yearly cost avoidance of 48.5 mCHF
  - 1 to 7 consolidation ratio on shared servers



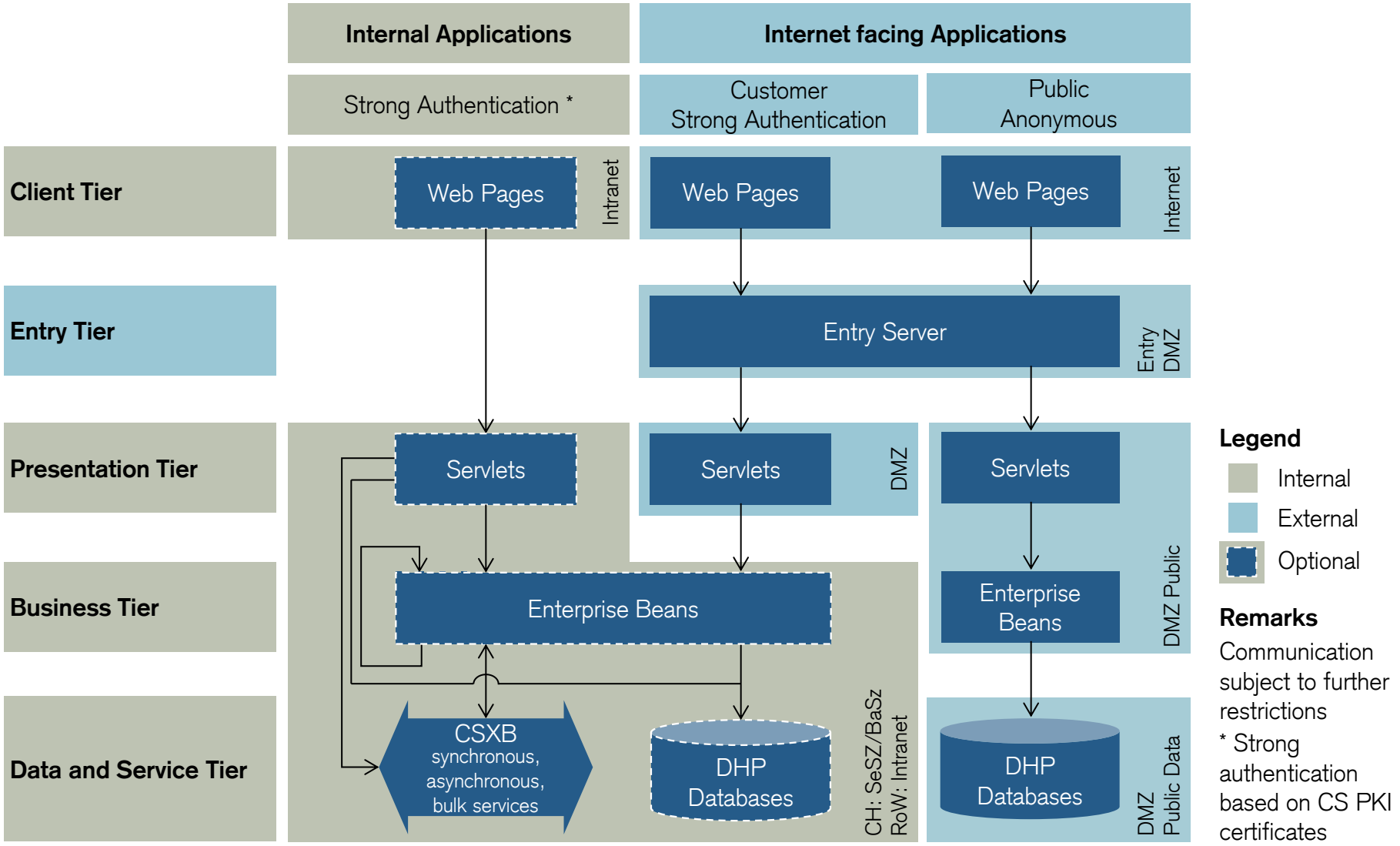


JAP Technical Stack (JAP AR 7)



* on exceptional basis

JAP Reference Architecture



Outline

1. Credit Suisse Platforms
2. Java Application Platform
3. **Our Vision: JAP in the Cloud**
4. A Customer's Perspective

General Cloud Characteristics*

Economy of Scale (larger workload, more variability)
- peak of the sum <= sum of the peaks
- aggregate demand typically smoother than individual
→ build for average demand rather than peak!

Clients

- self-service
- rent/return capacity
- pay as you go

Services

- rapid provisioning
- elasticity for applications
- measured usage

'cloud-like' services

CERTIFIED

Data Center

- process automation
- sharing/virtualization
- standardized machines & stacks

pooled, shared resources

by Amazon, Google, Microsoft, etc.

* **NIST:** High Scalability, Rapid Elasticity, Sharing and Multi-Tenancy, Measured Usage, Self-Service On-Demand

Cloud is *More than Virtualization!*
Impact for Application Development

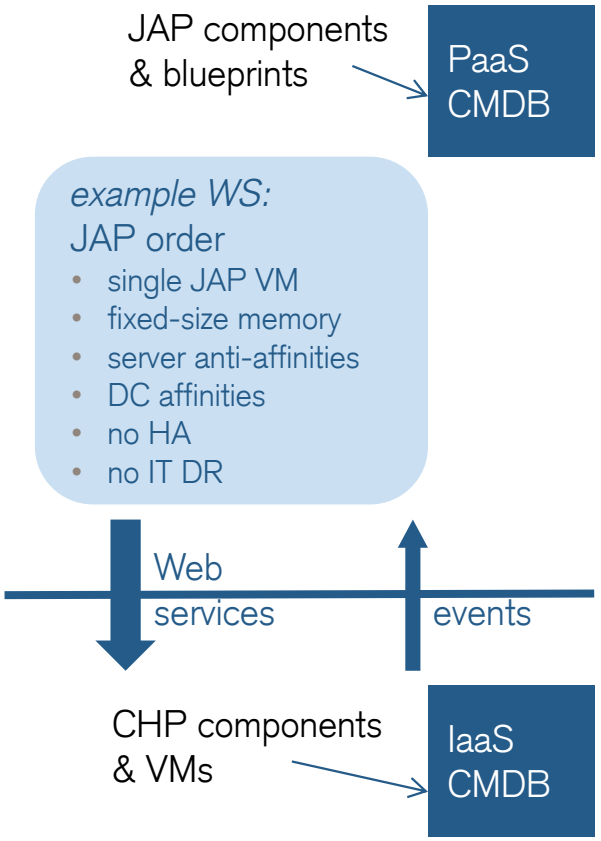
Cloud Aspect	Paradigm Shift for AD
Resource abstraction/simplification towards clients (compute, storage, network resources)	– Order capacity instead of HW – Choose from simple, standard options – Make no assumptions about placement (e.g. host names)
Rapid provisioning with self-service	– Test early, test often, explore – Test individually in entire application context – Rapid prototyping → early business feedback
Reproducible provisioning, configuration & deployment (persistent specifications with infrastructure service APIs)	– fully automatable 'on demand' test cycles (provision & build entire test env, run tests, decommission test env) – quickly reproduce production problems in UAT – exploit horizontal elasticity in production and maintenance
Rental model (pay as you go)	– Significantly lower entry cost (start small and quick) – Order and pay only what you need – Return what you currently do not need anymore
Elasticity (grow and shrink capacity on demand)	– Horizontal scalability – Statelessness – Fast startup, graceful shutdown of components

PaaS – Platform as a Service

Cloud for Development and Operation of Applications

PaaS **Application services based on high-level domain abstractions** **(reference architectures, container frameworks, progr. lang. artifacts, etc.)**	**IaaS** **Generic infrastructure services based on low-level HW/OS abstractions**
manage entire blueprints	manage single, unrelated VMs
rely on IaaS for capacity mgt → provide planning input to IaaS	capacity mgt of large server park → support different classes of reqs
horz. elasticity per application → capacity range as hint to IaaS	horz. elasticity within server park → minimize free pool
HA in middleware	HA on HW, hypervisor or OS level
IT DR in middleware	IT DR in hypervisor and storage
monitoring & logging for PaaS components	monitoring & logging for IaaS components (hypervisor, OS, …)
high-level measurements → # requests, E2E	low-level measurements → CPU, memory, I/O
manage PaaS CMDB items → blueprints & PaaS specific components	manage IaaS CMDB items → VMs & IaaS specific components

PaaS (e.g. JAP)



IaaS (e.g. CHP)

PoC Demo – Order/Provision JAP Blueprint

Use Case: Order Capacity in Dev/Test
• ***self-service*** • order ***entire blueprints*** along JAP reference architecture incl. presentation, business, and data tier ***interactively or programmatically*** • specify ***elasticity*** requirements • ***simplified*** specification choices only • ***very fast*** order fulfillment (seconds/minutes) • blueprint ***specifications*** can be stored/reloaded • **result**: number of virtual servers running order compliant JEE or DB stacks configured / interconnected according to reference architecture and ready for application component deployment

PoC Demo – Deploy Application to Blueprint

Use Case: Deploy Application Components in Dev/Test

• ***self-service***
• specify deployable ***application and configuration components*** per blueprint tier
• deploy components per tier or for entire blueprint
• ***very fast*** deployment and activation of components (seconds/minutes) ***interactively or programmatically***
• ***semi-automated component wiring***
  • automated remote call setup when possible
  • manual conflict resolution
  • manual specification for missing standards
• component ***specifications*** are ***reproducible*** (can be stored/reloaded); several deployable to same blueprint
• **result**: specified application components deployed to specified tier(s), wired up, and activated (ready to be used in application client requests)

Outline

1 Credit Suisse Platforms

2 Java Application Platform

3 Our Vision: JAP in the Cloud

4 A Customer's Perspective

Is Java EE ready for the Cloud?
A Customer's Point of View – Some Questions

- Is VM template cloning supported?
  - burnt-in assumptions about machine-specific configurations? (e.g. hostname hardwired in scripts)
- Is elasticity supported? (in general: dynamic automated blueprint management)
  - Growing: create and add instance to blueprint
    - rapid provisioning of a new instance (incl. application components) - e.g. VM template cloning helps
    - reproducible stack and configuration with own identity
    - after provisioning: startup of container and of application should be fast
    - KPI: how long does it take until new instance is ready to accept application requests
  - Shrinking: shutdown and remove instance from blueprint
    - block any new requests (not belonging to active sessions)?
    - session migration on demand? (some state might be hard to migrate to a different instance, e.g. distributed trx state or local cache)
    - other graceful starvation/shutdown measures? (incl. standard events to application?)
    - KPI: how long does it take until an instance's capacity can be given back?
  - Connection/environment management
    - can connection pools gracefully and transparently grow and shrink? (i.e. without application involvement)
    - load balancer, monitoring, log file mgt., etc. adjustable automatically?
  - Clustering
    - dynamic ad hoc adding/removing of an instance to/from a container cluster?
    - avoid cross-site clusters (operational independence!); avoid clustering anyway as much as possible (except for HA)?

Is Java EE ready for the Cloud?
A Customer's Point of View – Some Questions (cont.)

- Maintenance supported?
  - in place patching
    - guaranteed backwards compatibility
    - without reboot?
  - rolling upgrade (similar to elasticity!)
    - add new upgraded instance, then shutdown an old one - or the other way around
    - again: session migration on demand or other graceful shutdown measures?
- Resource management in shared environments?
  - heap size adjustable?
  - CPU and I/O requirements expressible by container? -> standard way to pass that to JVM and from there to OS and to hypervisor
  - in general: standard indication to hypervisors about memory usage for code that can be potentially shared across VMs