Building Applications for the Cloud: The Wrong, the Right, the Solid Way

( CloudCycle’14 – Stuttgart, Germany – September 22nd, 2014 )
Moving Images (The Wrong Way)
Packaging in a Virtual Machine

- Now, you package the whole application into a single virtual machine and move it to the cloud.
- Customers start using it from their browsers.
- They like it, and more and more are using it 😊.
- Thus, you need to scale!
Scaling Based on VMs

- You instantiate a second VM containing your application in the cloud
- Thus, your customers are happy!
- But, what about you?
  - How many licenses of App Servers, DBMS, CMS,… do you have to pay?
    - For example, if the customers use the Account features mostly, why do you replicate the Marketing stack and pay for the corresponding licenses?
  - What about your Account DB getting out of sync?
    - Storage is associated with single VM, but updates need to be synchronized across VMs to result in consistent data
Solving Scaling Related Problems: First Step

- You package the different stacks of your applications into separate VMs
- You persist your data in storage features of the cloud ("Data as a Service")
  - Data can then be shared when scaling out
- This enables replication of individual stacks for scaling
  - Avoiding the problems indicated before (licensing, data consistency,...)

Data as a Service (aspect of IaaS)
When a particular stack is under high request load, it can be scaled by starting multiple instances of the corresponding VMs.

- Data is shared between these VMs because database content is stored in storage features of IaaS.

- But maybe the underlying DBMS can sustain the load generated by many App Servers?
  - I.e. license cost can be reduced, etc.
Proper Granularity for Scaling

- You package “appropriate” components of your application in separate VMs so that they can scale independently.
- Now multiple VMs containing the App Server can use the same DBMS.
- But the DBMS in the separate machine needs maintenance.
- Do you want to do it by yourself?
Next step is to consider features provided by the cloud environment that may substitute components of your VMs

- For example, DBMS, App Server
  - E.g. Amazon SimpleDB, Google AppEngine,...
Towards “Cloud Native”

- Next, elasticity (i.e. on-demand scale-in & scale-out) requires...
  - Loose coupling of components
  - Automatic start/stop of instances of components
  - Stateless components
  - …
Loose Coupling: The Use of Queuing

Tight coupling (procedural programming)

Loose coupling (independent phases using queues)
Your application finally became
- Clusterable
- Elastic
- ...
Topology and Orchestration (The Right Way)
What We Understood So Far

- So, your building block is componentized
- You specify all middleware and infrastructure the application needs
- You specify all relations between these pieces and what the nature of that relations are
- You specified the **topology** of the application
A language for defining Service Templates …

… including a Topology Template describing the structure of a service

… including the definition of Plans for orchestrating the application

Definition of components of services

… along with the implementation artifacts of manageability operations

… and the definition of deployment artifacts of components

Definition of possible links between components

Packaging format for packaging models and all related artifacts.

OASIS Topology and Orchestration Specification for Cloud Applications

Cloud Service ARchive (CSAR)
A CSAR is an archive that makes your Cloud Application self-contained:
- it can be deployed and managed “everywhere”;
- it becomes a reusable artifact;
- it becomes tradeable;...
TOSCA TC Members (as of July 2014)

3M
ActiveState
Axway
CA Technologies
CenturyLink
Cisco
Citrix
Cloudsoft
EMC
Fujitsu
Google
HP
IBM
Huawei
Jericho Systems
NetApp
Nokia Siemens
Pricewaterhouse
Primeton
Red Hat SAP
Software AG
VCE
Vnomic
WSO2
Zenoss
Imperative vs Declarative Processing
A Sample Topology

- CustomerDB
- CustomerDB_on_MyDBMS
- MyDBMS
  - Install()
  - Attach()
- MyBlockStore_For_MyDBMS
- MyBlockStore
  - Allocate()
...And Its *Declarative* Processing

- In a declarative mode of processing, the environment does understand the specific processing requirements of all types
  - Node types
  - Relationship types
  - ...
- It further understands the dependencies of all these types
  - E.g. that hosted_on relationships must be processed before connected_to relationships

**PRO:** For provisioning and decommissioning, no plans need to be specified

**CON:** Very precise definition of all types and their dependencies must be specified
...And Its Imperative Processing

- In a imperative mode of processing, the environment is not dependent on a precise interpretation of the types
- All that interpretation is done by plans

PRO: No precise definition of all types, their processing, their behavior,... needed
CON: Plans must be specified even for “simple” provisioning and decommissioning needed
Deriving Plans from Topologies: The Basic Principle
Some More Details – At a Glimpse

Combined provisioning Dependency Graph

Provisioning Order Graph

Provisioning Flow


At a Glimpse: The Provisioning Subflows

\[ real(c) \in C_{provisionable} \lor p(c) = \text{multipleInst} \]

\[ p(c) = \text{multipleInst} \]

\[ p(c) \neq \text{multipleInst} \]

\[ real(c) \in C_{provisioned} \land p(c) \in \{\text{singleConfInst, singleInst}\} \]

\[ real(c) = \bot \land \implT(\impl(c)) \notin \{\text{providerSupplied, external}\} \]

At a Glimpse: Provision New Component Subflow

...and so on: the whole generation of “build plans” can be read in Ralph Mietzner’s PhD thesis 😊

Generating Management Plans

- This is more complicated!

...ongoing PhD Theses by Uwe Breitenbücher & Tobias Binz, IAAS, U of Stuttgart 😊
So...imagine you want to create an application that employs multiple of these technologies...
OpenTOSCA:
An Open Source TOSCA Environment
The OpenTOSCA Ecosystem

- TOSCA Modelling tool to develop CSARs
  - Graphical Topology Template Modeller
  - Template, Types, and Artifact Management Backend
The OpenTOSCA Ecosystem

- **TOSCA Runtime Environment**
  - Supports imperative processing based on BPEL
  - Supports Java and Script Implementation Artifacts
The OpenTOSCA Ecosystem

- **Winery**
  - Modelling Tool

- **OpenTOSCA Container**
  - Runtime Environment

- **Thek**
  - Self-Service Portal

- **TOSCA Runtime Environment**
  - Supports imperative processing based on BPEL
  - Supports Java and Script Implementation Artifacts

© Frank Leymann
The OpenTOSCA Ecosystem

- Self-Service Portal for OpenTOSCA
  - Provides easy graphical interface for users
  - Currently supports the provisioning of applications
The OpenTOSCA Ecosystem

winerary
Modelling Tool

OpenTOSCA Container
Runtime Environment

thek
Self-Service Portal

Develop
Deploy & Manage
Instantiate

„Standards-based - Open Source - End-To-End Toolchain Developed by IAAS at U of Stuttgart“
The OpenTOSCA Ecosystem

Winery
Modelling Tool

OpenTOSCA Container
Runtime Environment

Thek
Self-Service Portal

Develop

Deploy & Manage

Instantiate

„Standards-based - Open Source - End-To-End Toolchain Developed by IAAS at U of Stuttgart“
Vinothek

Moodle

Moodle bietet Ihnen virtuelle Räume zum Lernen und für Projektarbeit. Wir machen online Lernen einfach.

- Setzen Sie Texte, Bilder und Multimedia ein.
- Erstellen Sie Aufgaben und Tests selbst.
- Diskutieren Sie Inhalte und
- benachrichtigen Sie die Teilnehmer per E-Mail.
- Erarbeiten Sie gemeinsam Inhalte.
- Bilden Sie Gruppen.

Cloud Computing Patterns
(The Solid Way)
Your application finally became
- Clusterable
- Elastic
- ...

"Cloud Native" - Sketch
Native Cloud Application Properties - Exploiter View

- Clusterability
- Elasticity
- Multi-Tenancy
- Pay-per-Use
- Self-Service
Clusterability

- Application must be implemented as a distribute application
- Application consist of several types of components each of which can be instantiated multiple times, run concurrently and are loosely coupled
- Consequences:
  - Instances of a type of component may run on different machines
  - Number of instances may grow and shrink dynamically
Elasticity

- The application (more precisely: any of its components) should be able to scale up and to scale down – dependent on actual load

- Consequences:
  - Application should be stateless
  - A separate “watchdog” starts and stops instances of the applications component dynamically
    - Watchdog is steered by policies that specify conditions under which scale-up/scale-down should take place
Multi-Tenancy

- Application should be able to support isolated tenants
  - Application should be able to exploit multi-tenant middleware
- Note: Multi-tenancy does not mean isolation by associating each tenant with a separate copy of the application stack in one or more dedicated virtual machines (!)
Pay-per-Use

- This is enabled by granular metering and billing
  - The actual use of each component within the application stack must be able to be monitored, tracked, metered,...
- Tenant pays for use of application functions, not for provisioning, reservation,... of resources required by the application
Self-Service

- Each tenant can provision, manage,... “his” application on its own, whenever he decides to do so
  - No separate administrative staff (at cloud provider) is needed to the set the application up, configure it, run it, manage it
- Self-Service applies to each component of the application (including platform, infrastructure,...)
- Otherwise, there wouldn’t be real improvements in time-to-market
IDEAL Principle:
Native Cloud Application Properties - Architects View

**Isolated State:** most of the application is *stateless* with respect to:
- *Session State:* state of the communication with the application
- *Application State:* data handled by the application

**Distribution:** applications are decomposed to...
- ... use multiple cloud resources
- ... support the fact that clouds are large globally distributed systems

**Elasticity:** applications can be scaled out dynamically
- *Scale out:* performance increase through addition of resources
- *Scale up:* performance increase by increasing resource capabilities

**Automated Management:** runtime tasks have to be handled quickly
- Example: exploitation of pay-per-use by changing resource numbers
- Example: resiliency by reacting to resource failures

**Loose Coupling:** influence of application components is limited
- Example: failures should not impact other components
- Example: addition / removal of components is simplified
Best Practices How to Follow “The Solid Way”

Christoph Fehling · Frank Leymann
Ralph Retter · Walter Schupeck
Peter Arbitter

Cloud Computing Patterns
Fundamentals to Design, Build, and Manage Cloud Applications
Summary
Summary

- Capturing images of an application and bursting it to the cloud is the wrong way
  - You lose most benefits of the cloud
- Re-factoring your application into components explicitly specifying its use of middleware and infrastructure ("topology") as well as its management behavior ("orchestration") is the right way
- There are standards and Open Source implementations that ensure that an application can be run in multiple cloud environments
- Cloud Computing Patterns guide a proper structuring and management behavior to follow the best way
The End