Cloud Computing Patterns @ OOP 2017

Case Study and Discussion

http://www.cloudcomputingpatterns.org/oop17/

Dr. Christoph Fehling
christoph.fehling@daimler.com
Daimler AG

@ccpatterns
Mercedes Me
Part 1: Cloud Computing Patterns @ Mercedes Me

- What are the cloud computing patterns?
- What are the properties of a cloud-native application?
- How does a cloud-native application look like at Mercedes Me?

Part 2: the Non-technical “Stuff“ (Discussion)

- How does cloud computing affect procurement processes?
- Why are licenses of cloud products so problematic?
- How does cloud computing affect organizational hierarchies?
IDEAL Cloud Application Properties

**Isolated State**: most application components should be *stateless*. They do not handle:

- *Session State*: state of the communication with the application.
- *Application State*: business data handled by the application.

**Distribution**: cloud applications are split up into multiple components...

... to utilize multiple cloud resources.
... because the cloud itself is a large distributed system.

**Elasticity**: cloud applications are scaled by adjusting resource numbers (*scaling out*) – not by *scaling up*:

- *Scale out*: Increase performance by adding more resources.
- *Scale up*: Increase performance by improving existing resources.

**Automated Management**: management tasks during runtime have to be handled quickly.

Example: Cost reduction by adjusting pay-per-use resource numbers automatically.
Example: automatic reaction to resource failures.

** Loose Coupling**: application components should not influence each other regarding factors such as availability, data format, data exchange rate.

Example: failure of one application component does not cause failure of other components.
Part 1: Cloud Computing Patterns @ Mercedes Me
Workload Patterns @ Mercedes Me
Periodic Workload
IT resources with a peaking utilization at reoccurring time intervals experience periodic workload.

Once-in-a-Lifetime Workload
IT resources with an equal utilization over time disturbed by a strong peak occurring only once experience once-in-a-lifetime workload.

Unpredictable Workload
IT resources with a random and unforeseeable utilization over time experience unpredictable workload.

Continuously Changing Workload
IT resources with a utilization that grows or shrinks constantly over time experience continuously changing workload.

IDEAL Cloud-Native Application
- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling
Car Status Updates during a single Day

- **Peak workload**: beginning and end of each day
  → Rush hour
- **Low workload**: during each night
  → People are sleeping
Data Handling and Data Abstraction @ Mercedes Me
**Strict Consistency**
Data is stored at different locations to improve response time and to avoid data loss in case of failures while consistency of replicas is ensured at all times.

**Eventual Consistency**
Performance and availability of data in case of network partitioning are enabled by ensuring data consistency eventually and not at all times.

IDEAL Cloud-Native Application
- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling
**Strict Consistency**

\[ n \leq r + w \]

Number of replicas (\( n = 2 \))
- Replicas accessed to write (\( w = 2 \))
- Replicas accessed to read (\( r = 1 \))

**Eventual Consistency**

\[ n > r + w \]

Asynchronous update
- Failed update

IDEAL Cloud-Native Application
- **✓** Isolated State
- **✗** Distribution
- **✗** Elasticity
- **✓** Automated Management
- **✓** Loose Coupling
Data Abstractor
Data is abstracted to inherently support eventually consistent data storage through the use of abstractions and approximations.
Data Abstractor

### Cloud-Native Application

- **IDEAL**
- **Isolated State**
- **Distribution**
- **Elasticity**
- **Automated Management**
- **Loose Coupling**

### Data Abstractor

**Temperature Abstraction**
- Time Series: 3.14, 3.14159...

**Order Status Abstraction**

<table>
<thead>
<tr>
<th>Item</th>
<th>Packed?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
</tr>
<tr>
<td>3</td>
<td>no</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Parking Abstraction**

<table>
<thead>
<tr>
<th>Parking Lot</th>
<th>Free Slots</th>
</tr>
</thead>
<tbody>
<tr>
<td>City Center</td>
<td>16</td>
</tr>
<tr>
<td>Park &amp; Ride 1</td>
<td>86</td>
</tr>
<tr>
<td>Park &amp; Ride 2</td>
<td>102</td>
</tr>
<tr>
<td>Far Far Away</td>
<td>300</td>
</tr>
</tbody>
</table>
A lot of eventual consistent data:
- Odometer
- Fuel level
- Service intervals
- Errors shown in the car

If the connectivity of the car is limited (parking garage), an obsolete status is displayed!

But can all data be treated this way?
Mercedes Me App

Same as in the portal:
• Odometer
• Fuel level
• Service intervals
• Errors shown in the car

But certain interactions **must not be abstracted**!
• Door lock status
• Heating status
• Engine status (cooling)

→ Data abstraction and eventual consistency has to be evaluated for each data element!
Mercedes Me Microservice Template
**Node-based Availability**
A cloud provider guarantees the availability of nodes, such as individual virtual servers, middleware components or hosted application components.

**Environment-based Availability**
A cloud provider guarantees the availability of the environment hosting individual nodes, such as virtual servers or hosted application components.
Node-based Availability

Environment-based Availability

Five components are deployed (left): 0.9995^5 \approx 99.75%
**User Interface Component**

Synchronous user interfaces are accessed by humans, while application-internal interaction is realized asynchronously to ensure loose coupling.

---

**IDEAL Cloud-Native Application**

- Isolated State
- **Distribution**
- **Elasticity**
- **Automated Management**
- Loose Coupling

© Fehling
Cloud-Native Application

IDEAL

- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling

User Interface Component

Load Balancer

Request

ID

Number of Requests

User Interface Components

Message Queue

Elastic Load Balancer

scale

Mercedes Me Microservice Template
**Processing Component**

Processing functionality is handled by *elastically scaled components*.
**IDEAL**

Cloud-Native Application

- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling

**Processing Component**

- User Interface Component
- Elastic Queue
- Processing Components and other Application Components

- Request
- config
- Number of queued Messages
- scale

Mercedes Me Microservice Template

© Fehling
Lessons Learned

Moving from **Node-based Availability to Environment-based Availability** was the most challenging factor.

Cloud Computing is a significant enabler for **agile development**.

Soft factors (**Procurement, Licensing, Organizational Hierarchy**) pose a significant challenge!
Part 2: The Non-technical “Stuff”:
Procurement, Licenses, Organizational Hierarchies
1. Impact of Cloud Computing on Procurement Processes
Software scales with the number of cars as it is installed in each one of them.
Improvements / Fixes usually involve a visit to the shop.
Procurement of a Car Part Backend System

Change request (€€€) or new procurement (but only one supplier can do it): Vendor lock-in!

- User Behavior Change
- Cloud Interfaces
- New car models
- New functions
- Laws and regulations
- ...

Procurement processes have to reflect that a backend is not a car part!
*Procurement of cloud-based backend systems*

Cloud Properties (NIST)

- Self-Service Interface
- Rapid Elasticity
- Resource Pooling
- Measured Service

Cloud Customer (Supplier) - Manufacturer

- Flexible Use (Agile!) → Contract determines number of resources
- Cost Reduction → Limited, price possibly increased by supplier
- Pay for used resources → Pay for max. required resources

-> The Cloud Customer benefits from Cloud Properties. This should not be the supplier...

-> The Benefits of clouds are highly dependent on procurement Processes!

© Fehling
2. Licenses in the cloud
The License Conflict

Our product is the best to achieve these properties!

Vendor consultants

Vendor sales

Vendor consultants

Cloud Architects

Many component instances vs. cost reduction (= Monolith)

Procurement Department

-> Licensing models of many cloud products conflict with our architectural goals!
3. Impact of Cloud Computing on Organization Hierarchies
Organization Hierarchies and Clouds

Application Team

System A Responsible
Supplier A
Network ...

System B Responsible
Supplier B
DBMS ...

System C Responsible
Supplier C
VM ...

Ticket

UI Team

Application Team

Integration Team

Database Team

© Fehling 38
Organization Hierarchies and Clouds

- GUI Servlets
  - Web Server
    - User Accounts
    - Event Services
    - Billing
      - JEE Server
        - Spring Boot
          - MySQL DBMS
          - Key Value Storage
          - DB2

- Broker and Integration
  - UI Team
  - Application Team
  - Integration Team
  - Database Team

© Fehling 39
Side Note: Clouds Thrive on Automation!

- GUI
- Servlets
- Web Server
- User Accounts
- Event Services
- Billing
- JEE Server
- Spring Boot
- SAP
- MySQL DBMS
- Key Value Storage
- DB2

Amazon Cloud Formation

```json
"Resources": {
  "WebServerRole": {
    "Type": "AWS::IAM::Role",
    "Properties": {
      "AssumeRolePolicyDocument": {
        "Statement": [
          {
            "Effect": "Allow",
            "Principal": {
              "Service": [
                "Ec2Principal"
              ]
            },
            "Action": ["sts:AssumeRole"]
          }]
        },
        "Path": "/
      }
    }
  }
}
```

© Fehling
SIDE NOTE: CLOUDS THRIVE ON AUTOMATION!

Gui
Servlets

Web Server

User Accounts

Event Services

Billing

JEE Server

Spring Boot

SAP

MySQL DBMS

Key Value Storage

DB2

broker and integration

Docker Compose

---

Infrastructure as Code is mandatory to benefit from the cloud Properties!

© Fehling 41
OK! Let's Automate our IT Stack!

- **GUI Servlets**
- **WEB Server**
  - **User Accounts**
  - **Event Services**
  - **Billing**
- **JEE Server**
- **Spring Boot**
- **SAP**
- **MySQL DBMS**
- **Key Value Storage**
- **DB2**
- **Broker and Integration**

- **UI Team**
- **Application Team**
- **Integration Team**
- **Database Team**

© Fehling 42
OK! Let's Automate our IT Stack!

But Wait...

- Who creates (which part of) the model?
- Which System interprets and executes the model?
- Who controls the automation System?
- What about Existing Ticket Systems?!
- What about existing manual tasks?

--> Automation requires control over the Cloud to be centralized - not delegated!
--> Empowerment of DevOps Teams is needed to use Clouds efficiently!
--> Existing Teams may be afraid to loose control.
Lessons Learned

Revise procurement processes, because...

... suppliers using a cloud for you may create a vendor lock-in!

... many benefits of the cloud properties are lost if a supplier uses a cloud for you!

Demand cloud-compatible licenses, because...

... costs per instance conflicts with architectural goals!

Revise your organizational hierarchies, because...

... clouds thrive on automation and require fewer delegation of manual tasks!

... self-service interfaces are more agile than ticketing systems!

© Fehling 44
Summary
Part 1: Cloud Computing Patterns @ Mercedes Me

Microservice Template based on Cloud Computing Patterns and Pivotal Cloud Foundry

Summary

- Static Workload
- Periodic Workload
- Once-in-a-lifetime Workload
- Unpredictable Workload
- Continuously Changing Workload
- Strict Consistency
- Eventual Consistency
- Data Abstractor
- Node-based Availability
- Environment-based Availability
- User Interface Component
- Processing Component

© Fehling
Part 1: Cloud Computing Patterns @ Mercedes Me
Microservice Template based on Cloud Computing Patterns and Pivotal Cloud Foundry

Part 2: The Non-technical “Stuff”...

Procurement Processes have to be adjusted for cloud computing
- How can we buy environments for agile cloud development?
- How can we benefit from cloud properties - not our suppliers?

Organization Hierarchies have to be adjusted for cloud computing
- How can we benefit from cloud automation?
- How can we organize work without ticketing systems for manual tasks?

Part 3: Discussion during OOP 😊
I’m here all week! Contact me: fehling.c@gmail.com +49 170 58 35 456 @ccpatterns
http://www.cloudcomputingpatterns.org
Design Steps for Cloud Applications using Patterns

or

to see a Cloud Application Architecture you should go out and have a...
Design Steps for Cloud Applications

1. **Decomposition**
   - Workload
   - Data (State)

2. **Component Refinement**

3. **Elasticity and Resiliency**

---

IDEAL Cloud-Native Application

- **Isolated State**
- **Distribution**
- **Elasticity**
- **Automated Management**
- **Loose Coupling**

---

**Introduction**

© Fehling
How to distribute Application Functionality?
**Distributed Application**
A cloud application *divides provided functionality* among multiple application components that can be *scaled out independently*.

**Layer-based Decomposition**

Components reside on separate functional layers
- Often: user interface, processing, storage
- Access is only allowed to *same layer and the layer below*
  → Dependencies between layers and interfaces are controlled

**IDEAL Cloud-Native Application**
- ✔ Isolated State
- ✔ Distribution
- Loop Elasticity
- Loop Automated Management
- Loop Loose Coupling

© Fehling
**Distributed Application**

A cloud application divides provided functionality among multiple application components that can be scaled out independently.

**Process-based Decomposition**

Business process model determines decomposition

**Activities**: tasks executed in a specific order (**control flow**)

**Data elements**: information handled by activities (**data flow**)

Functional application components (**services**) are accessed by process

Cloud-Native Application

- **IDEAL**
  - Isolated State
  - Distribution
  - Elasticity
  - Automated Management
  - Loose Coupling

© Fehling
**Distributed Application**
A cloud application **divides provided functionality** among multiple application components that can be **scaled out independently**.

**Pipes-and-Filters-based Decomposition**

Decomposition based on the data processing function
- **Filter**: application component processing data
- **Pipe**: connection between filters (commonly messaging)

**IDEAL**

Cloud-Native Application
- ✔️ Isolated State
- ✔️ Distribution
- ☐ Elasticity
- ☐ Automated Management
- ☐ Loose Coupling

© Fehling
Coffe Shop – Decomposition of Functions

Identify functional components.

Distributed Application

Request Queue
- Drew (Latte)
- Chris (Coffee)
- Ben (Chai)
- Alex (Mocca)

Special Processing

Result Queue
- Eve (Tea)
- Frank (Frappu)

User Interface

Coffe Processing
What workload do components experience?
**Static Workload**
IT resources with an equal utilization over time experience static workload.

**Periodic Workload**
IT resources with a peaking utilization at reoccurring time intervals experience periodic workload.

**Once-in-a-Lifetime Workload**
IT resources with an equal utilization over time disturbed by a strong peak occurring only once experience once-in-a-lifetime workload.

**Unpredictable Workload**
IT resources with a random and unforeseeable utilization over time experience unpredictable workload.

**Continuously Changing Workload**
IT resources with a utilization that grows or shrinks constantly over time experience continuously changing workload.

---

**IDEAL Cloud-Native Application**

- Isolated State
- Distribution
- Elasticity ✔️
- Automated Management ✔️
- Loose Coupling
Coffe Shop – Workloads

Identify and compare workload generated by user groups at different components.

User Interface

Special Processing

Coffe Processing

Request Queue

© Fehling
Where does the application handle state?
**Stateful Component**
Multiple instances of a scaled-out application component synchronize their internal state to provide a unified behavior.

**Stateless Component**
State is handled external of application components to ease their scaling-out and to make the application more tolerant to component failures.

**Strict Consistency**
Data is stored at different locations to improve response time and to avoid data loss in case of failures while consistency of replicas is ensured at all times.

**Eventual Consistency**
Performance and availability of data in case of network partitioning are enabled by ensuring data consistency eventually and not at all times.

**Data Abstractor**
Data is abstracted to inherently support eventually consistent data storage through the use of abstractions and approximations.

---

IDEAL Cloud-Native Application

- **Isolated State**
- **Distribution**
- **Elasticity**
- **Automated Management**
- **Loose Coupling**

© Fehling
Coffee Shop – Data

Identify components storing data.
How are components implemented?
**User Interface Component**
Synchronous user interfaces are accessed by humans, while application-internal interaction is realized asynchronously to ensure loose coupling.

**Processing Component**
Processing functionality is handled by elastically scaled components.

**Batch Processing Component**
Requests are delayed until environmental conditions make their processing feasible.

**Multi-component Image**
Virtual servers host multiple application components that may not be active at all times to reduce provisioning and decommissioning operations.

**Message-oriented Middleware**
Asynchronous communication is provided while hiding complexity of addressing, routing, or data formats to make interaction robust and flexible.

**IDEAL Cloud-Native Application**
- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling

Component Refinement
Coffee Shop – Refinement of Components

Decide how to implement components.

Component Refinement

© Fehling
Elasticity and Resiliency
**Elastic Load Balancer**
The number of **synchronous accesses** to an elastically scaled-out application is used to determine the number of required application component instances.

**Elastic Queue**
The number of **accesses via messaging** is used to adjust the number of required application component instances.

**Node-based Availability**
A cloud provider guarantees the availability of nodes, such as individual virtual servers, middleware components or hosted application components.

**Environment-based Availability**
A cloud provider guarantees the availability of the environment hosting individual nodes, such as virtual servers or hosted application components.

**Watchdog**
Applications **cope with failures by monitoring and replacing** application component instances if the provider-assured availability is insufficient.

---

**IDEAL Cloud-Native Application**
- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling
Elasticity and Resiliency

What shall happen if workload changes or something fails?
Design Steps for Cloud Applications using Patterns

Decomposition

Workload

Data (State)

Component Refinement

Elasticity and Resiliency

IDEAL Cloud-Native Application

- Isolated State
- Distribution
- Elasticity
- Automated Management
- Loose Coupling

Summary
The End