Cloudiator: A Cross-Cloud, Multi-tenant Deployment and Runtime Engine (for IaaS)

Jörg Domaschka, Daniel Baur, Daniel Seybold, Frank Griesinger
OMI, University of Ulm, Germany
Disclaimer

Acknowledgements
The research leading to these results has received funding from the European Community’s Seventh Framework Programme (FP7/2007-2013) under grant agreement number 317715 (PaaSage).
Motivation

- Cloud hype is at its peak (or has even passed it)
- Still several huge problems around
  - Vendor lock-in
  - Uncomparable offerings (vCPU ≠ vCPU)
  - Incompatible APIs, unadopted standards
  - Cloud providers do not fit

- Need to adopt application and deployment to changing conditions
- Adaptation and re-deployment
- Awareness of application state and failures
Motivation (ii)

What is needed is a platform
- to provide multi- and cross-cloud capabilities
- to enable re-use of software components
- to provide an abstraction layer over different cloud APIs
- to support multi-tenancy
- to enact powerful adaptation rules

Cloudiator is such a tool
Scopes

- Deployment
- Runtime Handling
Deployment

Understanding of „application“

Understanding of „cloud“

Understanding of „lifecycle“
Understanding Applications

Component

• Self-contained chunk of software
• Unit of failure
• Unit of scale

• May interact with other components through *channels*

Examples

• Database
• Load balancer
• Web server/application server (in comb. with business logic)
Understanding Applications (ii)

Application

- Set of interdependent components
- Wired through channels

Diagram:
- Load balancer
- Application server
- Database
- Nginx
- Node.js & Ghost blog
- Postgres
Understanding Applications (iii)

Application Instance

- Enactment of an application in the "cloud"
- At least one component instance per component
- Definition of ports (if needed)
- Definition of locations (clouds and virtual machines)

```
+-----------------+    +-----------------+    +-----------------+
|     nginx       |    |      nodejs      |    |     postgres     |
|     8080        |  -->|    random        |  -->|     default      |
|                 |    |  nodejs + ghost blog |    |                 |
+-----------------+    +-----------------+    +-----------------+```
Deployment

Understanding of „application“

Understanding of „cloud“

Understanding of „lifecycle“
Understanding cloud terminology

**cloud platform**
- software stack
- version for
- management of (IaaS) resources
- defines API
- OpenStack Juno

**cloud provider**
- offers access to resources by running cloud platform
- defines endpoint (URI)
- Redstack, Uni Ulm cloud, bwCloud

**cloud**
- provider as seen by user
- user name
- access credentials
Dealing with Different Provider APIs

• Need for hiding the differences
• Cloudiator mostly relies on Apache jclouds
• … but has custom implementations as well
Technical Locations of bwCloud

cloud provider

OpenStack regions

availability zones
Geographical Locations of bwCloud

cloud provider

OpenStack regions

regionMA

regionUL

regionFR

‘real world’ location

GPS: … Mannheim

GPS: … Ulm

GPS: … Freiburg

Baden-Würt.

Germany

Europe

Germany

Europe

Baden-Würt.

regionFR

regionMA

regionUL

bwCloud

cloud provider

OpenStack regions

′real world′ location

GPS: … Mannheim

GPS: … Ulm

GPS: … Freiburg

Baden-Würt.

Germany

Europe

regionFR

regionMA

regionUL

bwCloud
Locations of Cloud Providers

But …

• availability zones may reside in different geographical locations
• other cloud platforms may use different schemas
• Not all locations are required for deployment
  (e.g. availability zone is optional)
Locations of Cloud Providers (ii)

Meta-model

- Top-level location
  - name
  - driver

- technical location
  - name
  - mandatory
  - geographical

parent

location hierarchy

OpenStack Juno

- "OpenStack Juno"
- jclouds/OpenStack

- "region"
  - true
  - required

- "availability zone"
  - false
  - optional
Other Properties

- locations, driver, and endpoints are cloud provider specific
- images and flavours are cloud-specific
  - different users may see different images/flavours
- same holds for virtual networks, security groups and the like
Cloudiator Users vs Cloud Users

Cloudiator user 'John Doe'

bwCloud access credentials

bwCloud provider

Amazon access credentials

Amazon provider
Adding a Cloud for a Cloudiator User

**Triggers harvesting**
- Available locations (regions and availability zones)
- Available images
- Available hardware configurations (flavours)
- Periodically updated
Images and Hardware Flavours

- **regionUL**
  - **bwCloud access credentials (doe)**
    - Image 43468ff6-
    - Image 72728324-
  - Hardware 1
  - Hardware 200
  - bwCloud access credentials (joerg)

- **regionUL**
  - 1 core, 2GB
  - 8 cores, 32GB
Operating System Hierarchy

Windows
- Windows 7
- Windows 8
- Windows Server 2012

Linux
- Ubuntu
  - 12.10
  - 14.04
  - 15.04
- RedHat
  - RHEL 6
  - RHEL 7

BSD
# Hardware Flavours

<table>
<thead>
<tr>
<th>id</th>
<th>localDiskSpace</th>
<th>memRam</th>
<th>numberOFcores</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5000</td>
<td>5000</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2048</td>
<td>2048</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>50000</td>
<td>50000</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>50000</td>
<td>50000</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>17450</td>
<td>17450</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2048</td>
<td>2048</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>4096</td>
<td>4096</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>512</td>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>16384</td>
<td>16384</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>102400</td>
<td>102400</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>4096</td>
<td>4096</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>8192</td>
<td>8192</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>17450</td>
<td>17450</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>17450</td>
<td>17450</td>
<td>3</td>
</tr>
</tbody>
</table>
Cloudiator components

UI
REST & Web

- users & accounts
- cloud providers
- deployment drivers (jclouds)
- flavours/images
- components & applications
Creation of Application Instances

• Creation of virtual machines
• 'Put' components on these virtual machines

• no magic
Creation of Application Instances (ii)
Deployment

Understanding of „application“

Understanding of „cloud“

Understanding of „lifecycle“
Lifecycle Handling

Lifecycle: defines actions to steer application component
• Fixed set of common handlers
  • install (download)
  • configure
  • start, stop
• Surveillance
  • start detector
  • stop detector
• Ports
  • downstreamUpdates
Clouditor Lifecycle Handlers (Example)
Technically Speaking …

Cloudiator (home domain)

- deploy
- installs
- creates

Lifecycle Agent

nginx

Docker container

virtual machine
Scopes

Deployment

Runtime Handling
Knowing what is going on …

• Judging your current deployment requires insight into the behaviour of
  • virtual machines
  • component instances
  • groups of component instances
  • …
• Monitoring is the key to this
Monitoring architecture (pt i)

Monitoring Agent
- monitors VM
- monitors container
- monitors application
  - default probes
  - custom probes
  - pull/push based
  - variable intervals
Dealing with Raw Data

- raw data is often useless
- at least aggregation is needed

→ collect data such that aggregation is possible
Monitoring architecture (pt ii)

Cloudiator (home domain)

Lifecycle Agent

Monitoring Agent

Docker container

nginx

Data Collector

Data Collector

- make data available to aggregators
- relay data if needed
- currently two implementations

- only has limited resources available

virtual machine

aggregators
# Where to aggregate?

Cloudiator’s rule of thumb: transmit as little data as possible

<table>
<thead>
<tr>
<th>scope</th>
<th>input from</th>
<th>aggregator at</th>
<th>output to</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>single vm</td>
<td>this vm</td>
<td>this vm collector</td>
<td>10 minutes CPU average of single container</td>
</tr>
<tr>
<td>cloud</td>
<td>vms in one cloud</td>
<td>any vm in cloud*</td>
<td>any collector in cloud</td>
<td>average of above across all instances of the same component</td>
</tr>
<tr>
<td>global</td>
<td>vms from at least two clouds</td>
<td>home domain</td>
<td>collector at home domain</td>
<td>average of above of all containers of cross cloud application</td>
</tr>
</tbody>
</table>
Cloudiator components
What to monitor and what to aggregate?

(Where do probes and aggregator configuration come from?)

• User defined
  • Investigate curves at GUI
  • User-requested data is also stored at home domain
• Part of the scalability rules definition …
Monitoring and Probing Example

```json
{
  "sensors": [
    {
      "name": "CPU",
      "type": "system.cpu",
      "interval": "1s"
    }
  ],
  "metrics": [
    {
      "name": "raw cpu",
      "scope": "blog.ghost.EACH",
      "type": "raw",
      "sensor": "CPU"
    },
    {
      "name": "avg cpu",
      "scope": "raw cpu.EACH",
      "type": "compute",
      "params": ["AVG", "10min", "raw cpu"]
    },
    {
      "name": "avg global",
      "scope": "SINGLE",
      "type": "compute",
      "params": ["AVG", "avg cpu.ALL"]
    }
  ]
}
```
PaaSage Scalability Rules Language (SRL)

• fine grained approach to specifying when to add new instances to a component
• Clouditor ships with an engine supporting SRL
• Basically same concept as for monitoring: conditions are treated as metrics
Scaling Rules Example

```json
{
    "rule":{
        "condition": ["AND",
                      ["avg cpu.ANY", "GT", "80%"],
                      ["avg global", "GT", "60%"],
                      ],
        "action": ["SCALE OUT",
                    {"scope": "component",
                    "target": "ghost"}
                    ]
    }
}
```
Cloudiator components (final view)

- Users & accounts
- Cloud providers
- Deployment drivers (jclouds)
- Data store
- Flavours/images
- SRL engine
- Components & applications
- Data collector
- Aggregators
Summary: What *Cloudiator* offers …

- simple application specification
- application instantiation (deployment)
- application operation (monitoring and adaptation)

- down to earth software suite
- with barely any magic

IaaS provider management

user management

open source, hosted@github

https://github.com/cloudiator
Our Roadmap

- Release of version 0.1 at beginning of August
  - Will have been tested by 4 PaaSage use cases by then
- For version 0.2 (end of September)
  - Finalise initial Windows support
  - Add more robustness (failure detection)
- For version 0.3 (end of 2015)
  - Add stateful migration of instances

- Introduce higher layers of abstraction
- Support further deployment mechanisms such as
  - Puppet, Chef, Dockerfiles