CloudsStorm: An Application-driven DevOps Framework for Managing Networked Infrastructures on Federated Clouds

Huan Zhou, Cees de Laat, Zhiming Zhao

University of Amsterdam

System and Network Engineering

Background

The IaaS (Infrastructure-as-a-Service) model in Cloud computing allows applications to customize its VM individually and configure its own network, but with limited programmability and controllability on the entire infrastructure. This gap hinders the application to customize its infrastructure at development phase and dynamic control at operation phase.

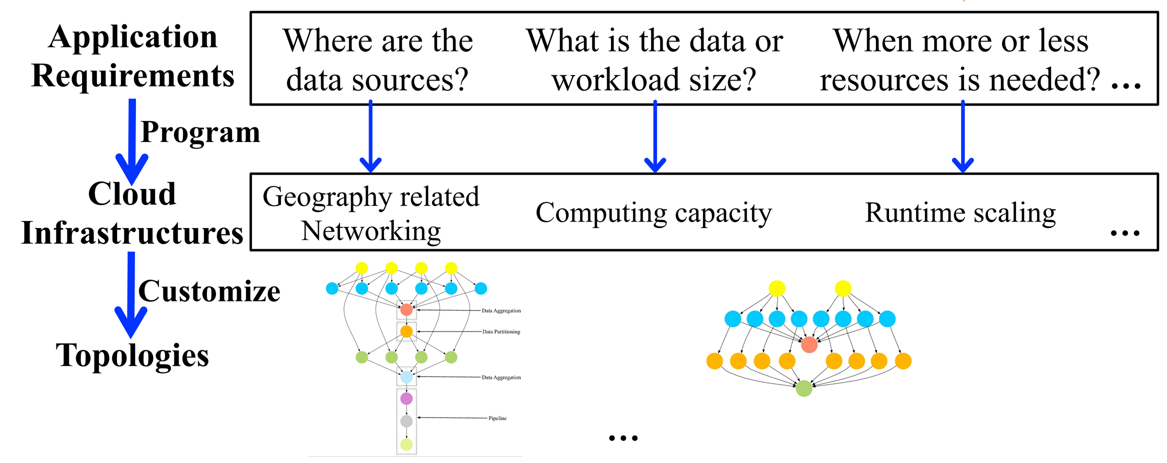
Research Problem and Challenges

Traditional DevOps (development and operations) approaches are not suitable for today's IaaS cloud environments, because of the slow, manual and error-prone collaboration between developers and operations personnel. The main challenges are as follows.

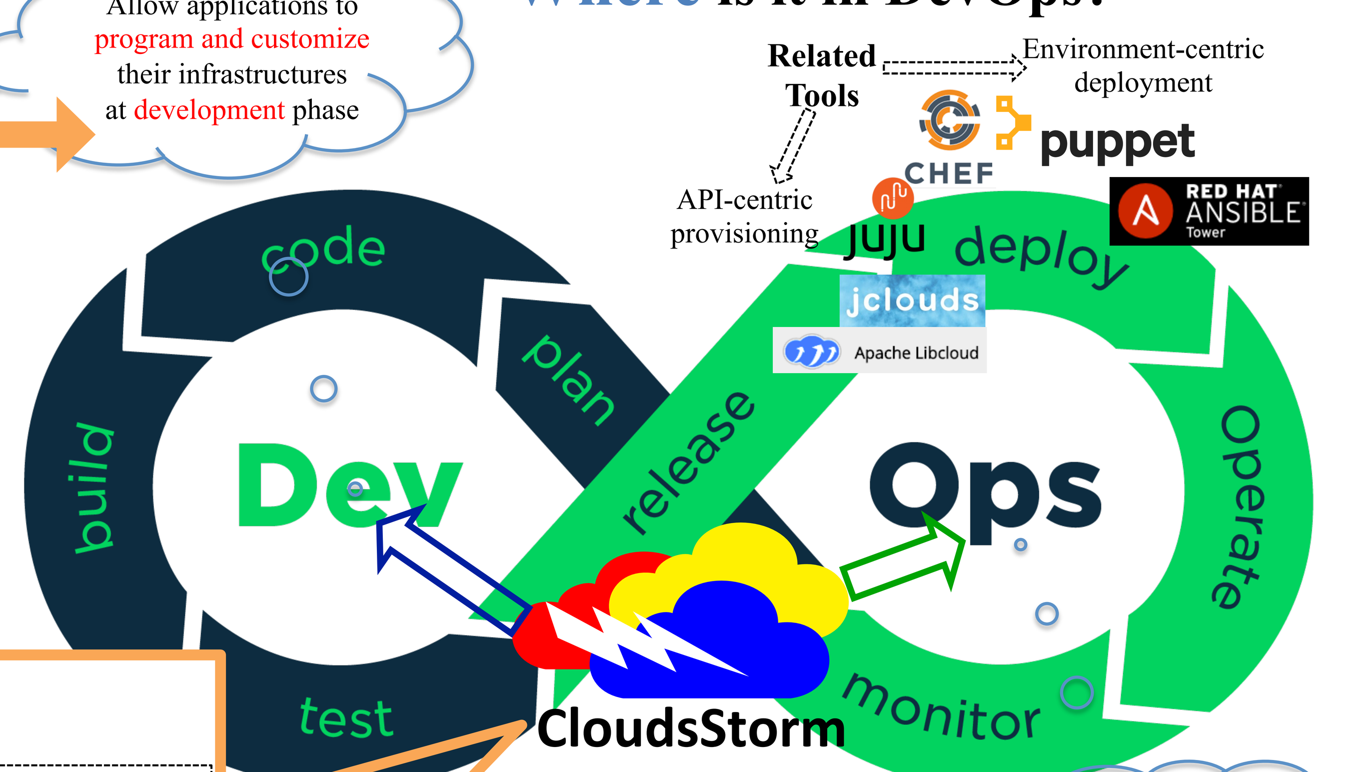
- Networked infrastructure: The network configuration for computing resources cannot be predefined, if adopting the unpredictable public IP addresses provided by Clouds. Most distributed applications rely on infrastructure with a customized network topology however.
- High-level controllability: There should be unified and application-driven controllability for the entire infrastructure, including failure recovery and auto-scaling, in order to ensure a high quality of service (QoS).
- Extensibility for federated Cloud: Clouds provide access to large quantities of computing resources. However, each Cloud has its own API for leveraging those resources. We therefore need an extensible framework to support different Clouds.
- Operation efficiency: Public Clouds adopt a pay-as-you-go business model. We need an efficient way to manage and operate these resources, with less manual work involved, in order to reduce monetary cost.

Overview

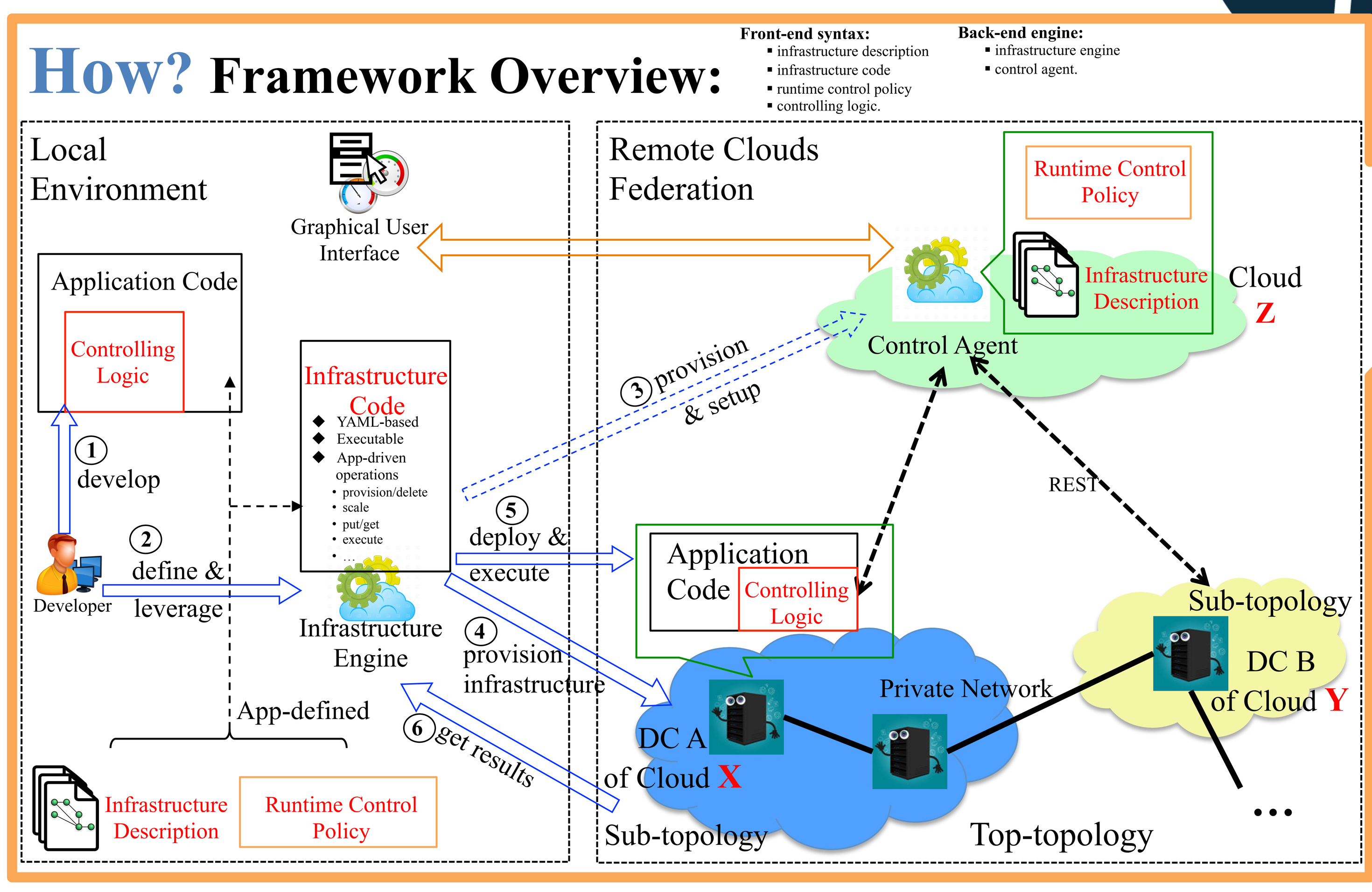
Why do we need it?



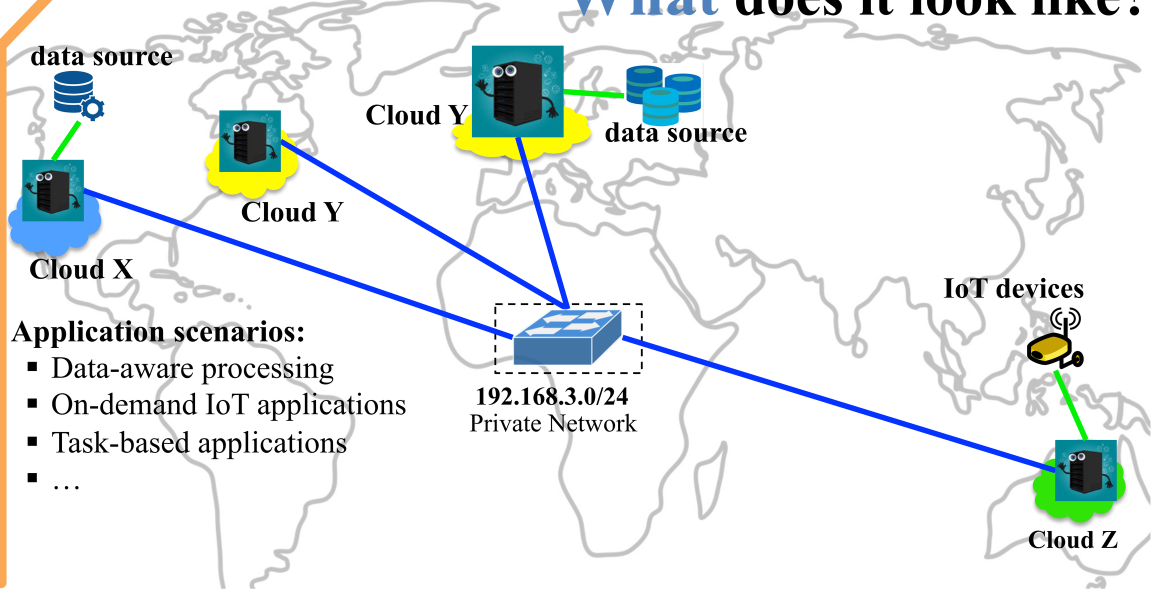
Where is it in DevOps?



How? Framework Overview:



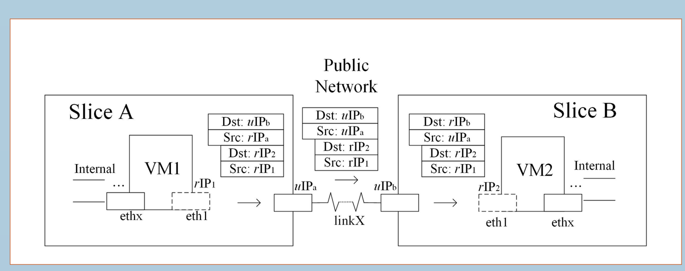
What does it look like?



Key Approaches and Implementation

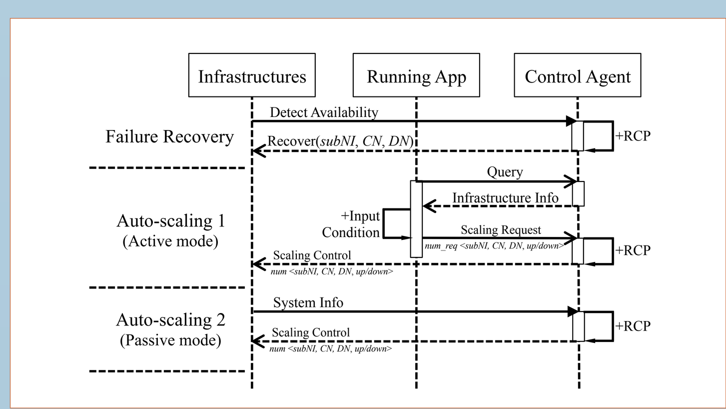
Networking

We adopt a tunnelling-based technique to realize the Virtual Network Function (VNF) between different Clouds.



Controlling Model

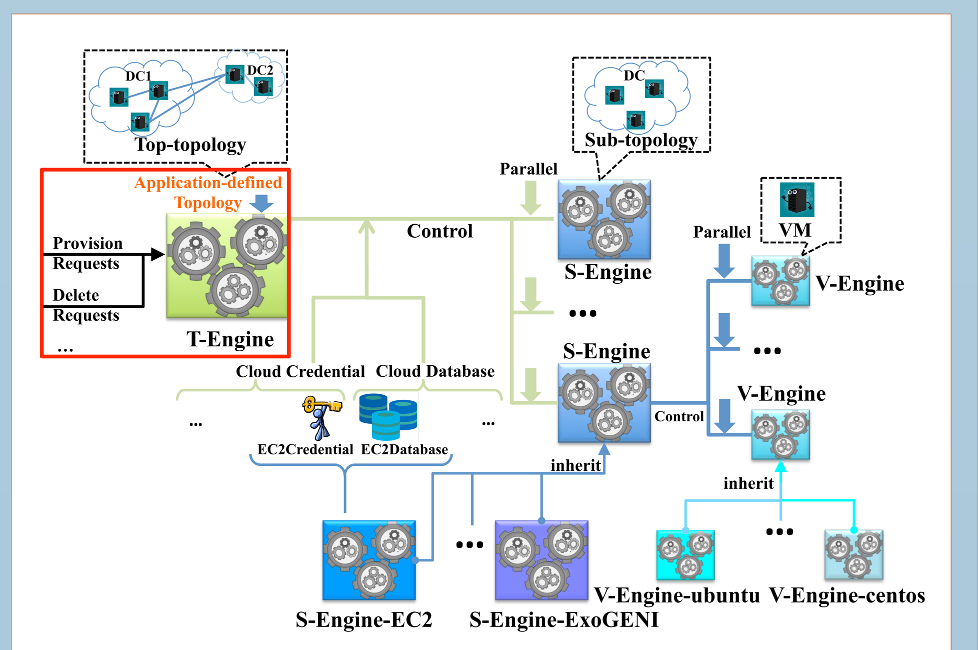
Our runtime controlling model supports multiple control scenarios.



Back-end Engine

TSV-Engine is the elementary engine used to control the infrastructure lifecycle.

T-Engine is responsible for top-topology management.
S-Engine is responsible for sub-topology management.
V-Engine is responsible for VM management.
The low-level engines are extensible and run in parallel.



YAML-based syntax

YAML (YAML Ain't Markup Language) is human-readable and easy to learn.
Topology Description Syntax (•) & Infrastructure Code Syntax (•)


```
Top-topology->
  userName: $User
  publicKeyPath: $Path
  topologies:
  - topology: $SubTopology
    cloudProvider: $Cloud
    domain: $DC
    status: 'fresh | running | deleted | failed'
  - *
  subnets:
  - name: $Subnet
    subnet: $subnet
    netmask: $netmask
    members:
    - vmName: $SubTopology.$Node
      address: $IP
    - *
```

```
Sub-topology->
  VMs:
  - name: $Node
    nodeType: $Type
    OSType: $OS
    script: $Path
  - *
```

```
Infrastructure-Code->
  - CodeType: 'SEQ | LOOP'
  - *
```

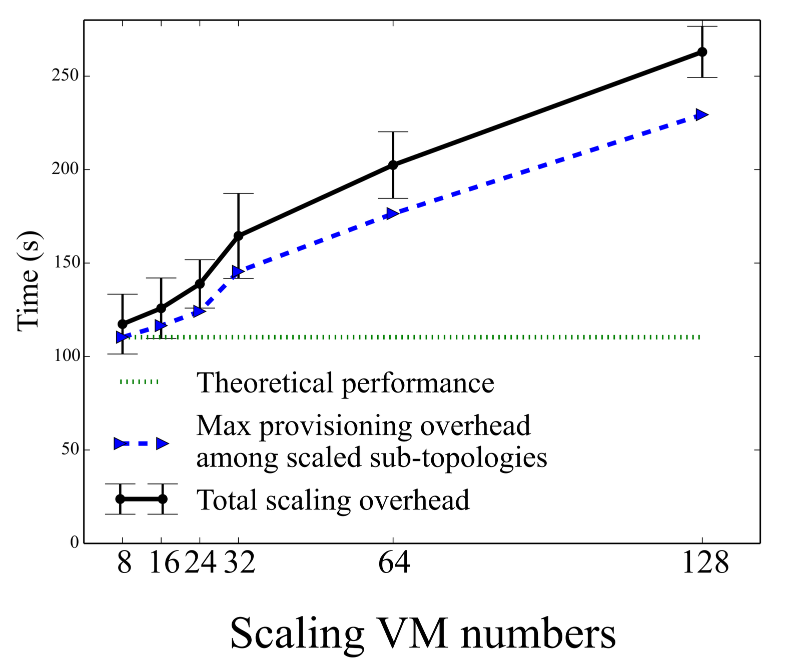
```
"SEQ"-Code->
  - CodeType: 'SEQ'
    OpCode:
      Operation: 'provision | delete | execute | put
  | get | vscale | hscale'
      [Options:]
      - [$String_i:$String_j]
      - *
      [Command: $String]
      ObjectType: 'SubTopology | VM | REQ'
      Objects: $Object_1 [||$Object_2]...
```

```
"LOOP"-Code->
  - CodeType: 'LOOP'
    [Count: $num]
    [Duration: $time_1]
    [Deadline: $time_2]
    OpCodes:
    - Operation: 'provision | delete | execute | put
  | get | vscale | hscale'
      [Options:]
      - [$String_i:$String_j]
      - *
      [Command: $String]
      ObjectType: 'SubTopology | VM | REQ'
      Objects: $Object_1 [||$Object_2]...
    - *
```

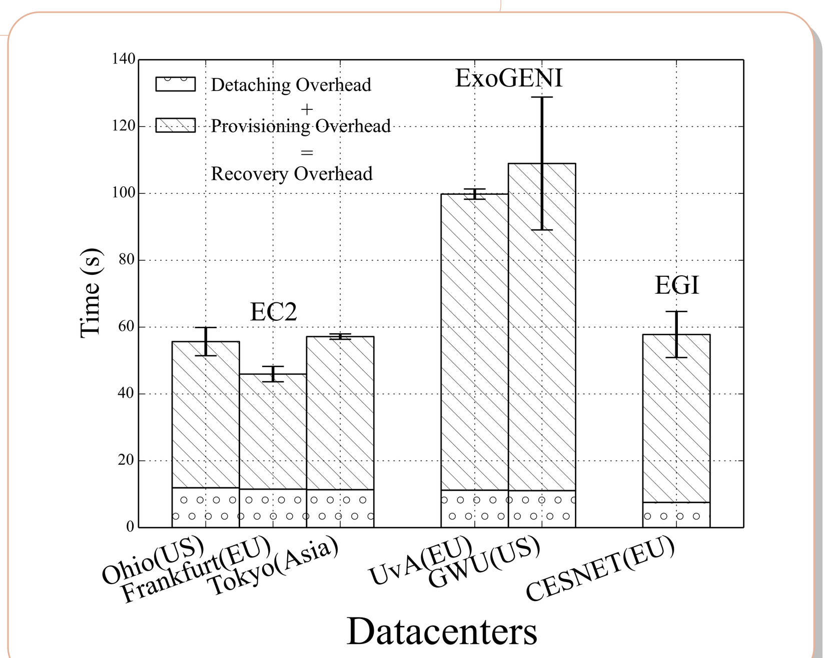

Experimental Results

Performance Evaluation

- **Scaling Performance**
- Test Cloud: ExoGENI
- VM type: XOMedium
- Scaling group: defined as 8 VMs in a sub-topology
- Scaling at different scales
- Each scaling group is from different datacenters

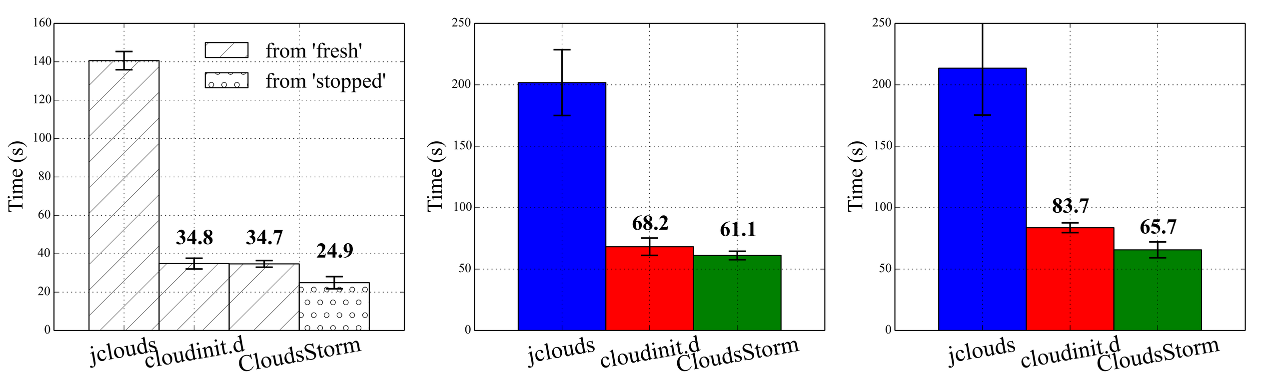


- **Recovery Performance**
- Test Clouds: EC2, ExoGENI and EGI
- VM Infomation:
  - CPU: 2 vCPU
  - MEM: 8 G
  - OS: Ubuntu 14.04
- Corresponding types defined in Clouds:
  - EC2: t2.large
  - ExoGENI: XOLarge
  - EGI: mem_medium



Performance Comparison

- Test Cloud: EC2 (California datacenter)
- VM type: t2.micro
- Scenarios:
  - (a) 5 VMs without deployment
  - (b) 5 VMs install Tomcat
  - (c) 4 VMs install Tomcat; 1 VM installs MySQL for database
- Comparison tools:
  jclouds (API-centric) VS cloudinit.d (Environment-centric) VS CloudsStorm (Application-driven)



(a) Scaling (b) Including Deployment (c) Deployment with Dependency

Contact: Huan Zhou (h.zhou@uva.nl)
Zhiming Zhao (z.zhao@uva.nl)

CloudsStorm: https://github.com/zh9314/CloudsStorm

Funded by:
SWITCH (643963), ENVRIPLUS (654182), VRE4EIC (676247).