a System for Management and Orchestration of Distributed Heterogeneous Cloud

Joacim.halen@ericsson.com
Initial Assumption

Create a cloud solution that leverages the network architecture Ericsson provides. Identify problems and verify solutions through prototypes.

Distributed Cloud
Distributed Heterogeneous Cloud
Distributed Heterogeneous Cloud

Big data center with $\sim 10^5$ servers
Distributed Heterogeneous Cloud

- Big data center with $\sim 10^5$ servers
- Small data center with $\sim 10^2$ servers
Distributed Heterogeneous Cloud

Each data center may run a different Cloud Operating System or stack, e.g. OpenStack, CloudStack, OpenNebula, etc.
Requirements and Design Goals

› Fully heterogeneous environment
› All APIs should be RESTful
› The system should be built around separate services
› Let applications drive requirements
› **Simplify** and **automate** as much as possible
Architecture (simplified)

› Separate services
› RESTful APIs
› Multi-tenant support
› Plug-in based
› Applications can use all APIs
Compute Service

**Extended with the concept of location**

› Geographical location
  - Region
  - Country
  - City
  - Data center (node)
    ‣ Rack
    ‣ Host

› Other
  - Latency
  - Close to IP
  - Between two nodes
  - At end of longest common path
  - Etc.
Simple network
Add context
Possible realization

- L2 tunnel
- GW
- L2 tunnel
- GW
- L2 tunnel
- GW

Erlang User Conference 2015 | Page 13
A Different context
Would Include elements as

› Firewalls
› NAT
› Routers
Problem

Server, storage, and network resources cannot be allocated independently of each other in a distributed cloud!
Solution

Separate resource allocation and placement from rest of resource management!
Container Service
Service Container (BNF)

BODY ::= \{"service" : { 
    "name" : STRING, 
    "vpcRef" : INTEGER, 
    "parameters" : { PARAMETERS }, 
    "definitions" : { DEFINITIONS }, 
    "temporals" : [ TEMPORALS ], 
    "scaling" : { SCALING_RULES }, 
    "networks" : [ NETWORKS ]\} \} 

DEFINITIONS ::= DEFINITION , DEFINITIONS  
               | DEFINITION

DEFINITION ::= NAME : OBJECT

OBJECT ::= SERVER | PORT | NETWORK
EX1 - specification

{
    "service": {
        "name": "Example 1",
        "definitions": {
            "S1": {
                "server": {… "Montreal" …}
            },
            "S2": {
                "server": {… "San Jose" …}
            },
            "S3": {
                "server": {… "Stockholm" …}
            }
        }
    },
    "networks": [
        {
            "network": {
                "layer": 2,
                "name": "Example Network",
                "attributes": {…},
                "ports": ["S1", "S2", "S3"]
            }
        }
    ]
}
}
Scaling Service
Scaling Service

• Based on set of application defined rules used as templates for how to add or remove infrastructure resources
• Defines limits on minimal and maximal amount of resources
• Application has full control on how to activate rules:
  • By using API calls
  • By defining automatic triggers specifying metrics to be monitored and thresholds to be met
Scaling Use cases

**No scaling** – application without scaling rules will not be auto-scaled.

**Application controlled scaling** – rules works as templates of possible complex infrastructure resources to be added or removed with one call from the application.

**Application defined automatic scaling** – rules will be invoked automatically by the trigger service using application defined triggers with specified metrics and thresholds.

**Application defined semi-automatic scaling** – rules will be invoked either by the application thru API or automatically by the trigger service using application defined triggers with specified metrics and thresholds, e.g. scale-out is monitored and triggered automatically and scale-in is triggered by application.
Scaling Rule (BNF)

SCALING_RULE ::= {
  "scaling-rule" : {
    "name" : NAME,
    "parameters" : { PARAMETERS },
    "initial_parameters" : IPARAMETERS,
    "scale_out" : SCALE-OUT,
    "scale_in" : SCALE-IN,
    "scale_up" : SCALE-UP,
    "scale_down" : SCALE-DOWN,
    "triggers" : [ TRIGGERS ],
    "template" : TEMPLATE,
    "notify" : [ RECIPIENTS ]
  }
}
Image Service
Implementation
A Closer Look

Compute Service

Adaptable Northbound Interface

Adaptable Southbound Interface
Compute Service

Northbound API

① YAWS receives the request
② YAWS creates a worker process
③ The worker calls out(…) in the API
④ Decode
⑤ Verify authorization
⑥ Translate from external to canonical
⑦ Dispatch to resource handler
⑧ Post process result
⑨ Translate from canonical to external
LocalToken = get_local_token(Tenant, Node),
case wpim:invoke(Node, ?WPIM_COMPUTE, server_create, [Node, LocalToken, Server, Flavor, Image]) of
...
Plug-ins

› Simple “behavior”

› Two callback functions

  load(Config) -> {ok, State}
  unload(State) -> ok

› All user defined functions that are exported must take an extra parameter “State”

  foo(P1, P2, State) -> {reply, Reply, Reply, State}

› Plug-ins can be defined to be pre-loaded or loaded at first use

› Plug-ins have a user defined type
PIM – Plug-in Manager

› Basic plug-in management
› Makes sure a plug-in is loaded when needed
› Thread safe, execution of user defined functions in a plug-in is done in the calling process, not in pim
› All calls to a plug-in is done through pim
   
   pim:invoke(Name, Function, Args)

› Finds plug-in based on name or type
› Search functions to find a plug-in or set of plug-ins
› More complex selection of plug-ins is done in wrappers
Wrappers

› **wpim** – Wind Plug-In Manager

› Location based selection of plug-ins

  wpim:invoke(Node, Name, Function, Args)

  wpim:invoke(Node, Type, Function, Args)

  wpim:invoke(NodeA, NodeB, Type, Function, Args)

  wpim:invoke(Name, Function, Args)

› **drim** – Driver Manager

› Singleton plug-ins, i.e. drivers

› Example, database driver
Evirt

› Erlang API to libvirt
› One-to-one mapping
› 280+ functions in API
› Supports libvirt 0.9.3
› Full support for callback functions
› Based on aspd
ASPD

Asynchronous Synchronous Port Driver

› Bridge between libraries
  – Erlang to C
  – C to Erlang
› Simple to use
› Support callback functions
› Library of convenience macros
› Support for logging
Testing

› Using eunit
› Tests at each level test that level and all levels involved below
› HTTP-client plug-in emulates a distributed OpenStack based cloud
› Wind does not know if it runs against a real cloud or the emulator
Testing

› Using eunit
› Tests at each level test that level and all levels involved below
› HTTP-client plug-in emulates a distributed OpenStack based cloud
› Wind does not know if it runs against a real cloud or the emulator
Reflection

Northbound request & response

- Most code handling a request executes in the worker process assigned by YAWS
- Request to internal processes are in most cases very short
- Less risk of deadlock in complicated chains

Southbound request & response
Current Work

› Fully distributed scheduler
› Policy description language and engine
› eflows
  – New behavior
  – Flows of tasks that will be executed as one
Q & A
Identity Service
Problem

Difficult to have global identities that spans over multiple data centers in a heterogeneous environment!
Example – Create Server 1

Tenant | Prime | Wind Compute | Keystone | Nova
--- | --- | --- | --- | ---
authenticate(…) | Global Token | server_create(GT, …) | authenticate(admin) | Local Admin Token
get_local_token(Tenant, Node) | Local Admin Token | tenant_create(LAT, …) | user_create(LAT, …) | Local Tenant
Local Tenant Admin User | authenticate(LTAU) | Local Token | Local Token | Local Token
Local Token | Local Server Object | server_create(LT, …) | |
Example – Create Server 2

Tenant

Prime

Wind Compute

Keystone

Nova

global server object

get_local_token(Tenant, Node)

Local Token

prime

server_create(GT, …)

local server object

server_create(LT, …)

Erlang User Conference 2015  |  Page 45
Code snippet

```erlang
LocalToken = get_local_token(Tenant, Node),
case wpim:invoke(Node,
    ?WPIM_COMPUTE,
    server_create,
    [Node, LocalToken, Server, Flavor, Image])
of
...
```