Enterprise Orchestration of Containers with Policies using TOSCA with Senlin, Heat-Translator and Magnum

Matt Rutkowski
IBM STSM, Cloud Open Technologies
OASIS TOSCA Chair, Simple Profile WG,
OpenStack Contributor: Heat-Translator, pyCASF

Julio Ruano
IBM Cloud, Open Technologies
OpenStack Contributor: Senlin, Heat-Translator
OpenStack Dojo Lead Instructor

#vBrownBag - Tuesday, October 27 @ 3:00pm
Senlin – clustering and policy service

A generic clustering service for an OpenStack cloud. Such a service is capable of managing the homogeneous objects exposed by other OpenStack components, such as Nova, Heat, Cinder etc. 1

A plugin-based:

- **Profile Management** enabling the creation and management of any object pools.
- **Policy Enforcement** framework featuring flexible policy customization for cluster management.
Senlin Container Clustering, AutoScaling, and LoadBalancing using Policies

Senlin Policy Types
- placement
- scaling
- deletion

Senlin works with existing OpenStack Services
- Magnum (Kubernetes / Docker)
- Heat
- Nova
- Ironic

Senlin Recognized Container Types
- Containers
- Stacks
- VMs
- BareMetal

Senlin Policy Types:
- health
- load-balance
- batching
Senlin – TOSCA Integration (Mitaka)

- Senlin Roadmap and Blueprints to normalize on the TOSCA Policy format
- Accept TOSCA format to express Clusters and Containers as TOSCA templates
- and work with the Heat-Translator service to parse, validate and deploy the templates

TOSCA v1.0 supports policies for **Placement, Scaling, Performance and Update**

Senlin will map TOSCA Policy “Triggers” to OpenStack Monitoring services
- Automatically register “Alarms” with choice of *Ceilometer, Monasca, Surveil*, etc.
- Tracks “State” of deployments / autoscales using performance & placement policies
Describing Linux Containers in TOSCA

using Docker as an example

**TOSCA Container.App Definition:**
a Node Template (Resource)

- **mysql_docker_resource**
- **Container.Application**

  - derived_from: tosca.nodes.Root
  - metadata: <tosca:map(string)>
  - version: <def_version_number>
  - description: <description>
  - artifacts:
    - mysql_docker
      - type: Image.Docker
      - URI: mysql
      - repository: docker
  - properties:
  - requirements:
    - host:
      - type: Container.Docker
    - storage:
      - type: Storage
    - network:
      - type: Endpoint

- **Requirements**
  - Container.Docker
  - Storage
  - Endpoint

**TOSCA Capability Definitions:**
(user to express requirements)

- **tosca.capabilities.Container.Docker:**
  - derived_from: tosca.capabilities.Container
  - properties:
    - num_cpus: <integer>
    - cpu_frequency: <scalar-unit.frequency>
    - disk_size: <scalar-unit.size>
    - mem_size: <scalar-unit.size>
  - attributes:
    - # Resource util. (value between 0-100%) which can be monitored.
    - utilization: <integer>

- **tosca.capabilities.Storage:**
  - # properties and attributes that define Storage properties
  - ...

- **tosca.capabilities.Endpoint:**
  - # properties and attributes that define storage requirements / mount path
  - # Allows 0..N port definitions
  - ...

**TOSCA Templates can model their repositories (e.g., Docker Hub)**

**Heat-Translator could dynamically “pull” templates, definitions, scripts, configuration files, etc. from multiple repositories**
**TOSCA Container Topology for Kubernetes**

*using the Kubernetes “redis” example (Kubernetes Pod, Containers)*

---

**Cluster Policy**

**Cluster Scaling Policy**
- Event = Magnum
- Condition = Capacity > 80%
- Action = scale()

---

**Pod (TOSCA Group)**

**Containers (TOSCA Nodes)**

- **redis-master-pod**
  - Kubernetes.Pod
  - type: tosca.groups.Placement
  - sources:
    - master-container
    - sentinel-container

**“MemberOf” Relationship (implied)**

---

**Container Policy**

**Container Scaling Policy**
- Event = Magnum
- Condition = Capacity > 80%
- Action = scale()
Senlin can use TOSCA Heat-Translator to map to Kubernetes

for use with Magnum (Container Service)
TOSCA Policies – Allows Triggers to be declared based upon an **Event, Condition, Action** model

**Event**
- Name of a normative TOSCA Event Type
- That describes an event based upon a Resource “state” change.
- Or a change in one or more of the resources attribute value.

**Condition**
Identifies:
- the resource (Node) in the TOSCA model to monitor.
- Optionally, identify a Capability of the identified node.
- Describe the attribute (state) of the resource to evaluate (condition)

**Action**
Describes:
- An Operation (name) to invoke when the condition is met
- within the declared Implementation
- Optionally, pass in Input parameters to the operation along with any well-defined strategy values.

**TOSCA Policy Definition:**

```xml
<policy_name>:
  type: <policy_type_name>
  description: <policy_description>
  properties: <property_definitions>
  # allowed targets for policy association
  targets: [ <list_of_valid_target_resources> ]
triggers:
  <trigger_symbolic_name_1>:
    event: <event_type_name>
    target_filter:
      node: <node_template_name> | <node_type>
      # (optional) reference to a related node
      # via a requirement
      requirement: <requirement_name>
      # (optional) Capability within node to monitor
      capability: <capability_name>
      # Describes an attribute-relative test that
      # causes the trigger’s action to be invoked.
      condition: <constraint_clause>
    action:
      # implementation-specific operation name
      <operation_name>:
        description: <optional description>
        inputs: <list_of_parameters>
        implementation: <script> | <service_name>
```

...
Applying a TOSCA Scaling Policy to Resources in a Cluster using Senlin and the Kubernetes “redis” example (Pod, Docker Containers)

**TOSCA Scaling Policy Definition**

```yaml
my_scaling_policy:
  type: tosca.policies.scaling
  properties:  # normative TOSCA properties for scaling
    min_instances: 1
    max_instances: 10
    default_instances: 3
    increment: 1
  # target the policy at the “Pod”
  targets: [redis-master-pod]
  triggers:
    # Symbolic trigger name
    resize_compute:
      event: tosca.events.resource.utilization
      target_filter:
        node: master-container
        requirement: host
        capability: Container
        condition: utilization greater_than 80%
      action:
        # map to SENLIN::ACTION::RESIZE operation
        resize:
          # operation name
          inputs:
            strategy: BEST_EFFORT
          # Invoke SenLin webhook (service)
          Implementation: Senlin.webhook()
```

**Normative Properties**
- TOSCA provides base properties used for scaling

**Trigger Name**
Could be used to reference an externalized trigger

**Monitor Resource**
- Find Resource (node) named “master-container” in TOSCA topology.
- Locate its “host” Container
- Attach an alarm | alert | event to identified Container resource and monitor for Condition

**Action**
- Invoke “resize” operation
- Within Senlin “webhook” (service)

**Parameters**
- optionally provide input parameters, for example “strategy” with value of “BEST_EFFORT”

**Policy Target**
- In this case, a Kubernetes Pod
- A TOSCA resource type (tosca.groups.placement type)

**TOSCA normative event type** (name) that would map to domain-specific names (e.g., OpenStack Ceilometer)

**Map Event, Condition to target Monitoring Service**
- Map event to OpenStack notifications for resource (e.g., Nova)
- Register alarm with the target monitoring service (e.g., Ceilometer, Monasca, etc.)

Based upon the Senlin “scaling_out_policy.ceilometer.yaml”
Example: Senlin Scaling Policy mapping to TOSCA

**Senlin policies are helping shape the TOSCA v1.1 Policy definition**

### Senlin Definition

```
type: ScalingPolicy
version: 1.0

schedule:
  start_time: "2015-05-07T07:00:00Z"
  end_time: "2015-06-07T07:00:00Z"

alarm:
  type: SENLIN::ALARM::CEILOMETER
  properties:
    meter: cpu_util
    op: gt
    threshold: 50
    period: 60
    evaluations: 1
    repeat: True # always true

handlers:
  - type: webhook
    action: SENLIN::ACTION::RESIZE
    params:
      type: CHANGE_IN_CAPACITY
      number: 1
      strategy: LEAST_USED

  credentials:
    user: john
    password: secrete
  - type: email
    addresses:
      - joe@cloud.com
```

### TOSCA Definition

```
my_scaling_policy:
  type: tosca.policies.scaling
  version: 1.0
  properties:
    min_instances: 1
    max_instances: 10
    default_instances: 3
    increment: 1
  targets: [redis-master-pod ]
  triggers:
    resize_compute:
      event:
        type: tosca.events.resource.utilization
        implementation: Ceilometer
        target_filter: ...
        condition: utilization greater_than 50%
  schedule:
    start_time: "2015-05-07T07:00:00Z"
    end_time: "2015-06-07T07:00:00Z"
```

### Version

- TOSCA v1.1 feature

### Schedule

- TOSCA v1.1 feature

### Alarm

- Could add an "implementation" field to TOSCA event keyname.

### Condition

- Fields map to TOSCA "condition" section

### Sampling

- TOSCA v1.1 will add "collection"

### Handler

- Describe in the TOSCA policy "action:" section
- number == increment

### Credentials

- Security design flaw. Should never be in a policy definition.

Senlin source: “scaling_out_policy ceilometer.yaml”
Example: Senlin Placement Policies in TOSCA

Senlin WIP Definition

```yaml
# Sample placement policy doing round-robin
type: senlin.policy.placement
version: 1.0
description: A policy for node placement scheduling.
properties:
  availability_zones:
    # Valid values include:
    # ROUND_ROBIN, WEIGHTED, SOURCE
    strategy: ROUND_ROBIN
    candidates:
      - AZ1
      - AZ2
  regions:
    # Valid values include:
    # ROUND_ROBIN, WEIGHTED, SOURCE
    strategy: ROUND_ROBIN
    candidates:
      - RegionOne
      - RegionTwo
```

TOSCA Definitions *(Treat each policy independently)*

```yaml
my_zone_placement_policy:
  type: tosca.policies.placement
  properties:
    container type: availability_zone
    containers: [AZ1, AZ2]
    targets: [allowed_nodes]
  triggers:
    placement_trigger_zone:
      event: tosca.events.resource.schedule
      target_filter: <node/group to be placed>
      condition:
        action:
          schedule: # operation name
          inputs:
          strategy: ROUND_ROBIN
          weighting_map: { [AZ1: 2], [AZ2: 1] }
          implementation: Senlin.webhook()
```

```yaml
my_region_placement_policy:
  type: tosca.policies.placement
  properties:
    container type: region
    containers: [RegionOne, RegionTwo]
    targets: [allowed_nodes]
  triggers:
    placement_trigger_region:
      event: tosca.events.resource.schedule
      action:
        schedule: # operation name
        inputs: # optional input parameters
        strategy: ROUND_ROBIN
        implementation: Senlin.webhook()
```

+ Aggregate

```yaml
my_zone_placement_policy:
  type: tosca.policies.placement
  properties:
    container type: availability_zone
    containers: [AZ1, AZ2]
    targets: [allowed_nodes]
  triggers:
    placement_trigger_zone:
      event: tosca.events.resource.schedule
      target_filter: <node/group to be placed>
      condition:
        action:
          schedule: # operation name
          inputs:
          strategy: ROUND_ROBIN
          weighting_map: { [AZ1: 2], [AZ2: 1] }
          implementation: Senlin.webhook()
```

= matching container(s)

Zone and Region are separate Policies

- Customers wish to **describe** and manage (placement) policies **independently** of each other (e.g., based upon security, based upon logical location, etc.).
- **Associate** at deployment time
- Each can be **aggregated** by policy service (as they have the same “target(s)”)

Senlin Source: [placement_rr.yaml](#)
Example: Senlin Load Balancing Policies in TOSCA

Senlin Definition

```
# Load-balancing policy spec using Neutron LBaaS service

type: senlin.policy.loadbalance

version: 1.0

description: A policy for load-balancing the nodes in a cluster.

properties:

  pool:
    # Protocol used for load balancing
    protocol: HTTP
    # Port servers are running on the members
    protocol_port: 80
    # Name or ID of port subnet
    subnet: private-subnet

  lb_method: ROUND_ROBIN

session_persistence:

  # valid values include: ROUND_ROBIN, LEAST_CONNECTIONS,
  # SOURCE_IP

  type: SOURCE_IP

vtep:
  # Name or ID of address Subnet
  subnet: public-subnet
  # IP address of the VIP
  address: <ADDRESS>

  # Protocol used for VIP
  protocol: HTTP
  # TCP port to listen on
  protocol_port: 80

  # Max connections/second allowed for VIP
  connection_limit: 500
```

TOSCA Definition (Senlin Type)

```
my_load_balancer_policy:
  description: Senlin policy for load-balancing

# define a new SenLin policy type, as TOSCA does not
# have a normative one for Load Balancing:

type: senlin.policy.LoadBalance

properties:

  strategy:
    # well-known strategy values in TOSCA
    ROUND_ROBIN | LEAST_CONNECTIONS | SOURCE_IP

  pool:
    # Reuse TOSCA Endpoint for “pool”, “vip”
    # It already has protocol, port (and ports),
    # network_name, etc. properties

    # Endpoint(s) of application (internal)
    pool_endpoint: <tosca.types.Endpoint>

    # (Virtual) Endpoint of application (external)
    virtual_endpoint: <tosca.types.Endpoint>

  session_persistence:
    # valid values include:
    # SOURCE_IP, HTTP_COOKIE, APP_COOKIE

    type: SOURCE_IP

  vtep:
    # Name or ID of address Subnet
    subnet: public-subnet
    # IP address of the VIP
    address: <ADDRESS>

    # Protocol used for VIP
    protocol: HTTP
    # TCP port to listen on
    protocol_port: 80

    # Max connections per second allowed
    connection_limit: 500

    targets: [<node/group to be blanced>]
```

TOSCA is looking to create a normative Group subclass for Cluster

- include load balancing and HA requirement descriptions
- In order to describe these as part of the Application Topology.
Please join us as a contributor!

Senlin project:
https://wiki.openstack.org/wiki/Senlin

Heat-Translator project:
https://wiki.openstack.org/wiki/Heat-translator

Exploring Magnum and Senlin integration for autoscaling containers

Ton Ngo, Julio Ruano, Qi Ming Teng

Thursday, 9:50am Kyokko