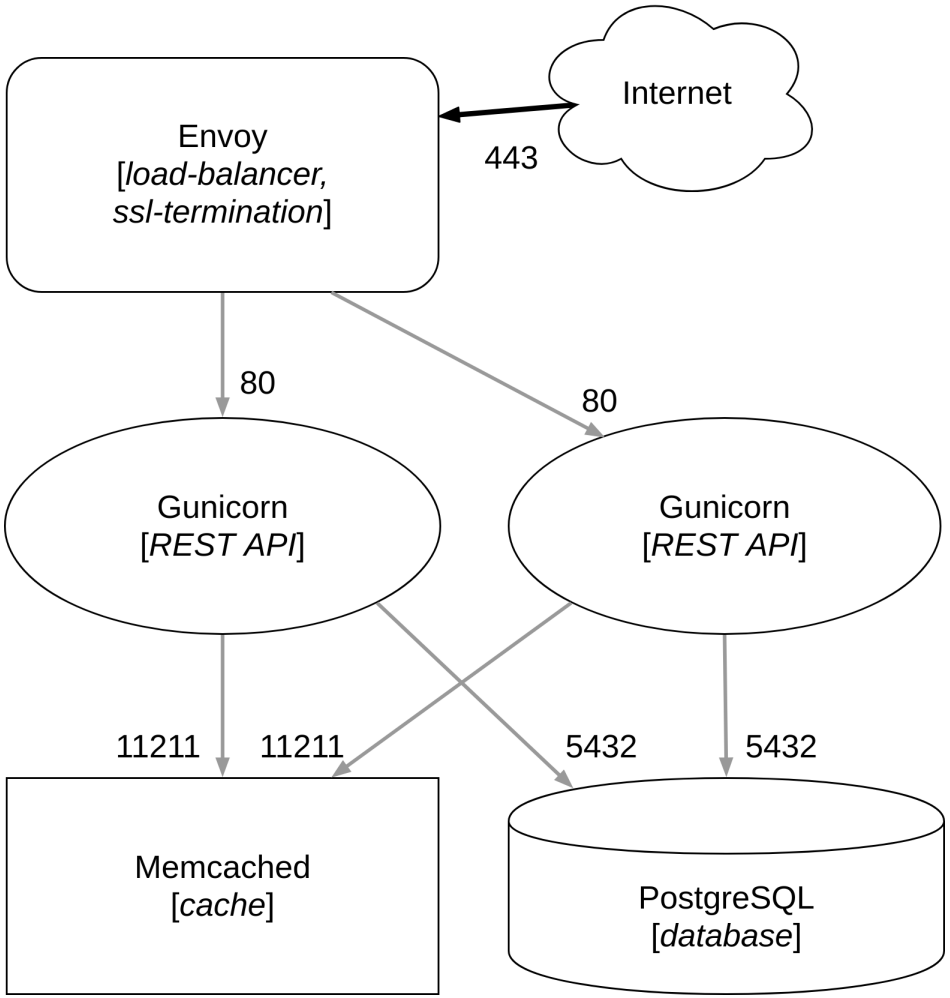
Domain-specific language for infrastructure as code

Valeriya Shvetcova, Oleg Borisenko, Maxim Polischuk

13 september 2019

* This work is funded by the Minobrnauki Russia
(grant id RFMEFI60417X0199, grant number 14.604.21.0199)

Introduction: infrastructure example



Infrastructure requirements:

- Envoy:
  - public IP
  - CPU count
- Gunicorn:
  - individual requirements
- Memcached:
  - large RAM
- PostgreSQL:
  - reliable block device

Introduction: state-of-the-art

Existing methods of infrastructure deployment:

- *manually* using cloud user interface (UI)
- program scripts using *multicloud libraries*: Fog, etc.
- *system orchestrator*: CloudFormation, Heat, Ansible, etc.

Ansible is an open source system orchestration tool

- manages either cloud resources, or physical servers
- easy to develop because of modularity

High resource demand of methods above when **migrating infrastructures** into/between cloud environments is the main problem

The goal

The goal is to provide declarative language for **unified description** and **deployment** of infrastructures **without reference to any cloud proivder**.

Infrastructure description:

- IP addresses
- connected networks and ports
- node parameters, such as
  - operating system
  - CPUs number, amount of RAM and external memory
  - open ports

TOSCA standard

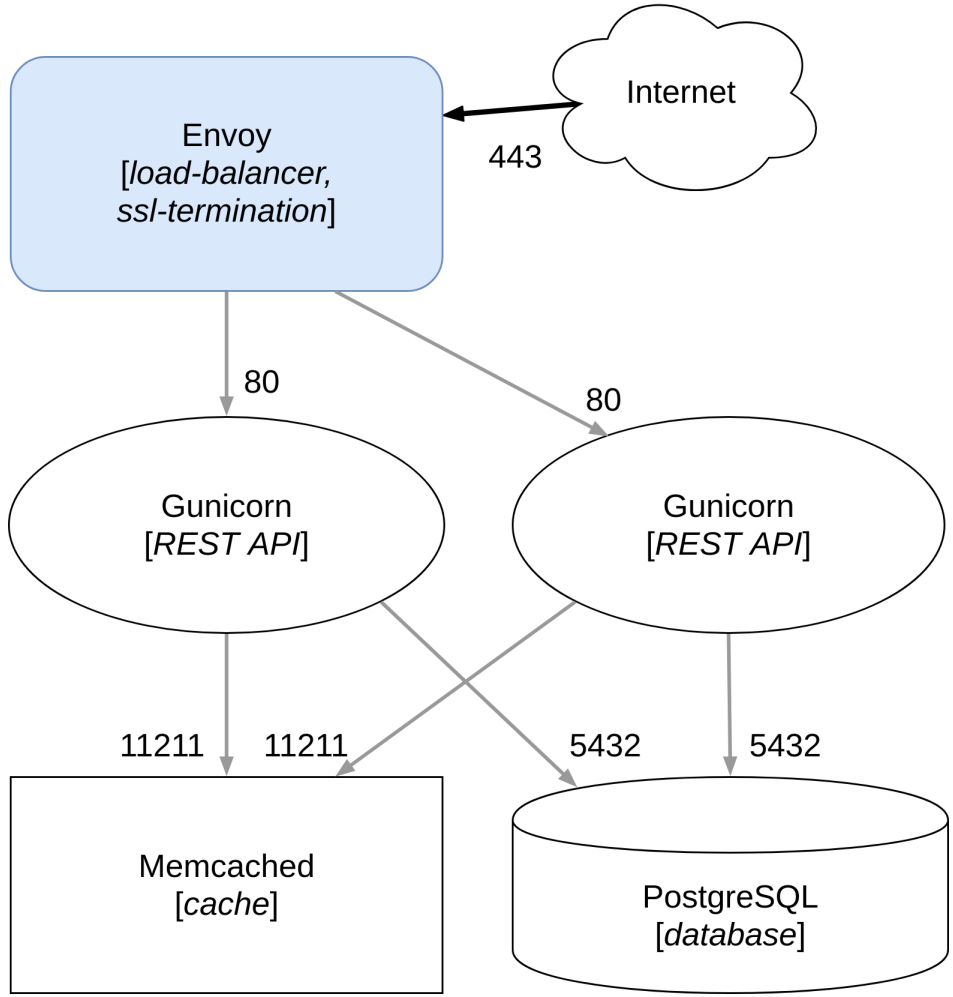
TOSCA specifies template language for defining required topology of a software application in the cloud and its orchestration process

Important capabilities of TOSCA и OCCI **standards**:

- **unified** infrastructure model for different cloud providers
- description of **physical** and **virtual** resources in clouds
- **high level abstraction** of infrastructure components

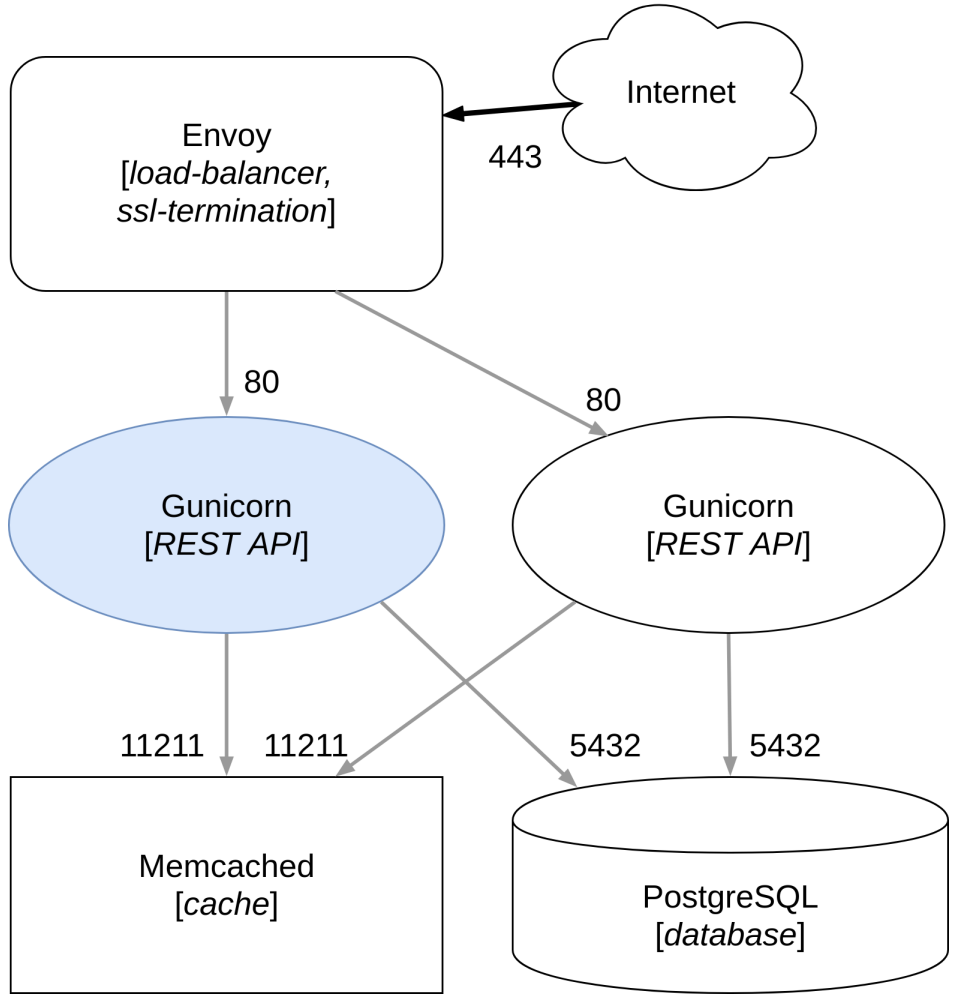
Example of using TOSCA standard for infrastructure description

```
envoy_load_balancer:
  type: tosca.nodes.Compute
  attributes:
    public_address: 10.10.0.1
    networks:
      - network_0:
          network_name: internal
  capabilities:
    host:
      properties:
        num_cpus: 16
    endpoint:
      properties:
        protocol: tcp
        port: 443
        initiator: target
      attributes:
        ip_address: 0.0.0.0/0
    os:
      properties:
        type: ubuntu
        version: 16.04
        architecture: x86_64
        distribution: xenial
```



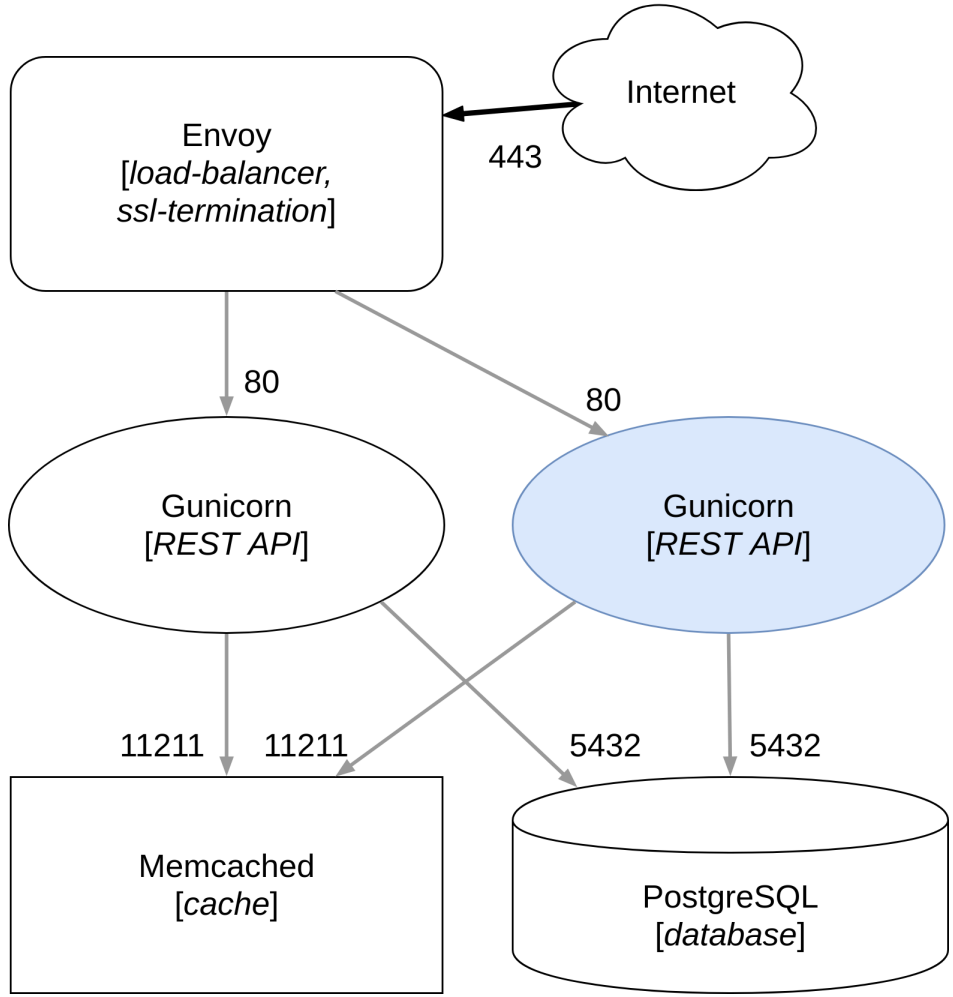
Example of using TOSCA standard for infrastructure description

```
gunicorn_rest_api_0:
  type: tosca.nodes.Compute
  attributes:
    networks:
      - int:
          network_name: internal
  capabilities:
    host:
      properties:
        num_cpus: 16
    endpoint:
      properties:
        protocol: tcp
        port: 80
        initiator: target
      attributes:
        ip_address: 192.168.112.0/24
    os:
      properties:
        type: ubuntu
        version: 16.04
        architecture: x86_64
        distribution: xenial
```



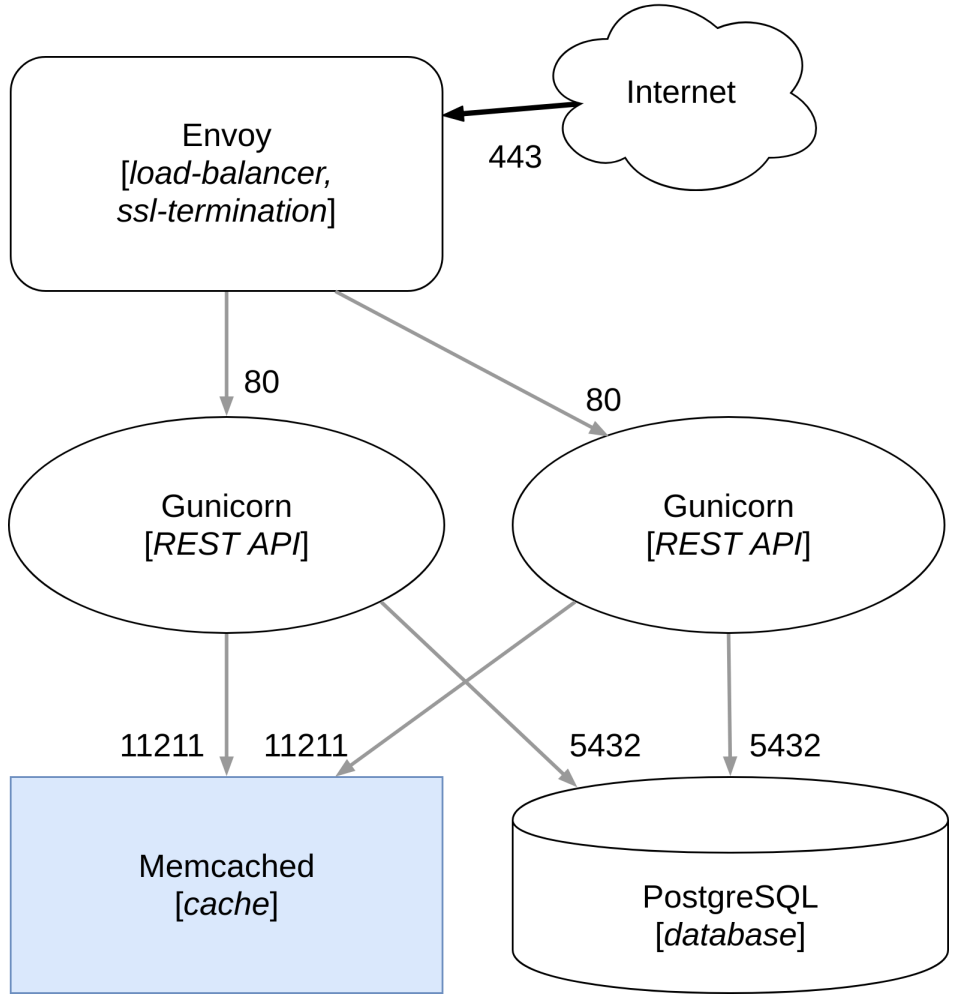
Example of using TOSCA standard for infrastructure description

```
gunicorn_rest_api_1:
  type: tosca.nodes.Compute
  attributes:
    networks:
      - int:
          network_name: internal
  capabilities:
    host:
      properties:
        num_cpus: 16
    endpoint:
      properties:
        protocol: tcp
        port: 80
        initiator: target
      attributes:
        ip_address: 192.168.112.0/24
    os:
      properties:
        type: ubuntu
        version: 16.04
        architecture: x86_64
        distribution: xenial
```



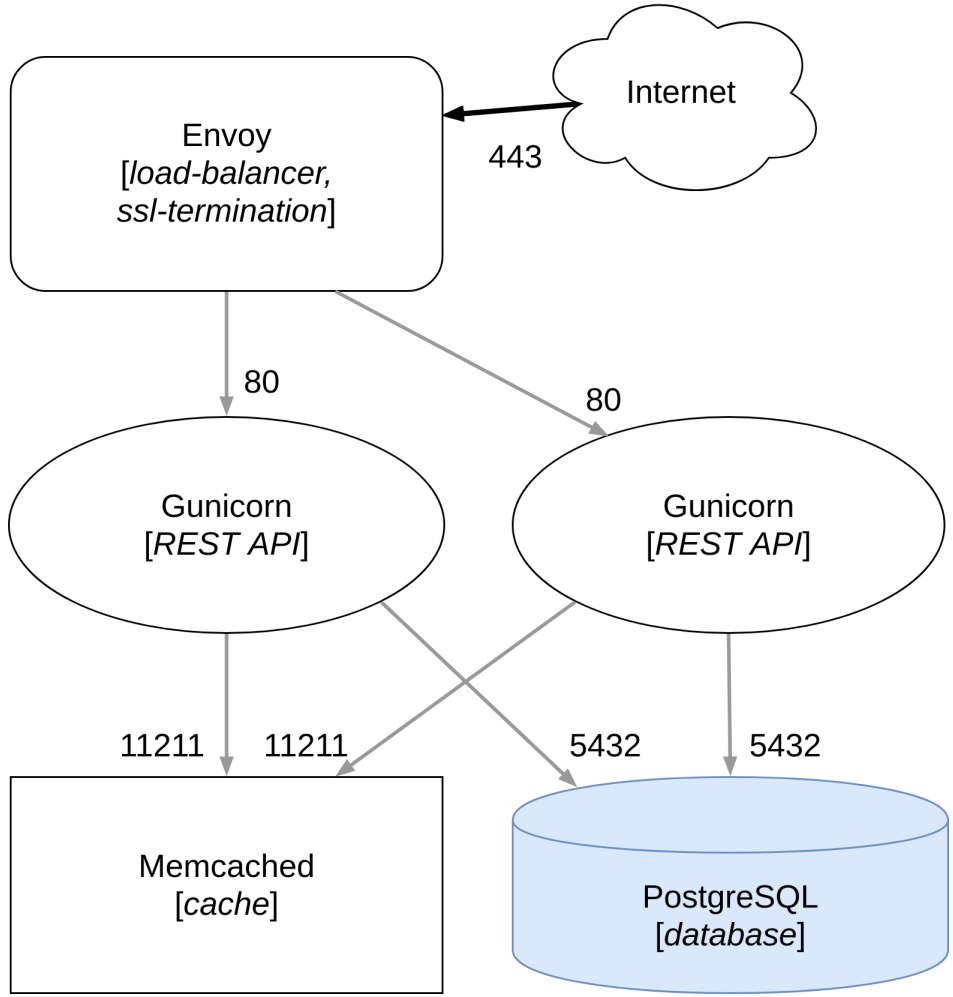
Example of using TOSCA standard for infrastructure description

```
memcached_cache:
  type: tosca.nodes.Compute
  attributes:
    networks:
      - int:
          network_name: internal
  capabilities:
    host:
      properties:
        mem_size: 64 GiB
    endpoint:
      properties:
        protocol: tcp
        port: 11211
        initiator: target
      attributes:
        ip_address: 192.168.112.0/24
    os:
      properties:
        type: ubuntu
        version: 16.04
        architecture: x86_64
        distribution: xenial
```



Example of using TOSCA standard for infrastructure description

```
postgresql_database:
  type: tosca.nodes.Compute
  attributes:
    networks:
      - int:
          network_name: internal
  capabilities:
    host:
      properties:
        disk_size: 1024 GiB
    endpoint:
      properties:
        protocol: tcp
        port: 5432
        initiator: target
      attributes:
        ip_address: 192.168.112.0/24
    os:
      properties:
        type: ubuntu
        version: 16.04
        architecture: x86_64
        distribution: xenial
```



Interpreter algorithm

TOSCA interpreter is developed as Ansible module:

- **Input:** TOSCA template with service topology description and cloud provider name
- **Steps:**

Step 1 translate TOSCA template into the Ansible tasks
Step 2 implement generated Ansible playbook

- **Output:** generated Ansible playbook that deploys the required service in required cloud

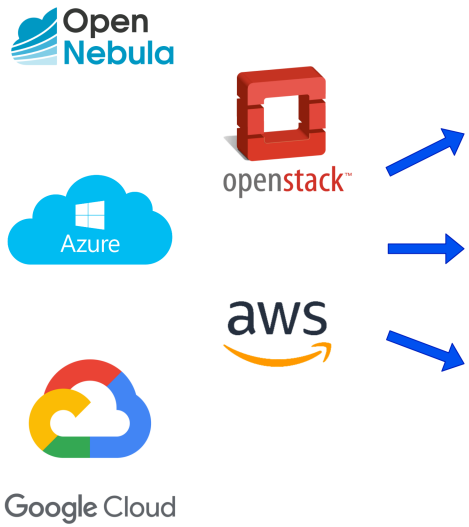
Interpreter input (left side) and output (right side)

```
envoy_lb:
  type: tosca.nodes.Compute
  capabilities:
   host:
     properties:
       num_cpus: 16
    os:
     properties:
        type: ubuntu
        version: 16.04
        architecture: x86_64
        distribution: xenial
  attributes:
   networks:
     - network_0:
         network_name: internal
   public_address: 10.10.0.1
```

```
- name: Create Server
  os_server:
    name: envoy_lb
   flavor: cpuonly.medium
   image: Ubuntu 16.04 <...>
   nics:
   - internal
- name: Create FloatingIp
  os_floating_ip:
    floating_ip_address:
                10.10.0.1
    network: ispras_net
    server: envoy_lb
```

Adding new cloud provider support

Developed interpreter is **independent** of any cloud provider.
Adding a provider support process consists of three files:



- define the used cloud resource facts in Python
- define the Ansible cloud module as a special TOSCA entry in YAML-file
- define mapping between TOSCA parameters and Ansible module arguments

Defining Ansible cloud module as a special TOSCA entry: example

```
openstack.nodes.SecurityGroupRule:
  attributes:
    direction:
      type: string
      default: ingress
      constraints:
        - valid_values: [ egress, ingress ]
    protocol:
      type: string
      constraints:
        - valid_values: [ tcp, udp, icmp, 112, None ]
      required: false
    <...>
  requirements:
    - security_group:
        node: openstack.nodes.SecurityGroup
        <...>
```

Defining mapping between TOSCA parameters and Ansible module arguments: example

```
tosca.nodes.Compute.capabilities.host.properties.num_cpus:
    parameter: openstack.nodes.Server.requirements.
        flavor.node_filter.properties.vcpus
    value: {self[value]}

tosca.nodes.Compute.attributes.networks.*.addresses:
    parameter: openstack.nodes.Server.requirements.
        nics.node_filter.properties.id
    value:
        value: network_id
        facts: openstack_subnet_facts
        condition: ip_contains
        arguments
          - allocation_pool_start
          - allocation_pool_end
          - {self[value]}
```

Conclusion

Basic results:

- Interpreter of language specified by TOSCA standard was developed independent of any cloud provider
- Interpreter provides easy process adding new cloud provider support

Discovered problems:

- nonexistence of complete mapping between cloud provider resources
- lack of support even basic parameters of standard by most advanced cloud providers

Future work

- Full support for OpenStack, Amazon and other cloud providers
- Automate adding cloud support: Ansible facts and modules definition
- Support some Ansible variables (specify rules and abilities)
- Validate mapping between TOSCA parameters and Ansible module arguments
- Support for PaaS level

Thank you for attention

Introduction: definitions

DSL

Domain-specific language (DSL) is a programming language specialized for a particular use.

IaC

Infrastructure as Code (IaC) is a model aimed to describe an infrastructure in a declarative format using a high level abstraction of infrastructure components.

IT Infrastructure

IT infrastructure is a combination of hardware, software, network resources and services necessary for deployment, maintaining and managing the application environment.

Interpreter usage

Example of using developed Ansible module — `cloud_infra_create_by_tosca`

```
---
- name: Create server infrastructure with OpenStack
  hosts: all
  tasks:
    - name: Create infrastructure using TOSCA template
      cloud_infra_by_tosca_create:
        cloud: ispras
        template_file: tosca-server-example.yaml
        provider: openstack
        facts: "{{ ansible_facts }}"
```

Existing projects using TOSCA

- **OpenTOSCA**
  - only **Winery** is developed, graphical web-based tool for modeling TOSCA topologies
- **TOSCA2Chef**
  - is not available open-source
- **Cloudify**, **Indigo DataCloud**
  - relies on specific cloud provider