Orchestration Tool Roundup
Howdy

I am Uri Cohen

Head of Product @ GigaSpaces, makers of Cloudify

@uri1803
My (Other) Most Important Project
Agenda

- Orchestration 101
- Method of comparison
- Tool overview
Orchestration 101
Orchestration 101

- **Common Characteristics**
  - Use DSL to define “blueprint”
  - Execute a process based on input from the blueprint
  - Pass context information between the deployed entities

- **Different assumptions lead to different approaches**
  - Application Architecture
  - Infrastructure
  - Scope of automation
Method of Comparison

- Same application requirements
- “Production grade” deployment
- Broken into three main groups
  - Pure Play – Cloudify/TOSCA, Terraform
  - Infrastructure Centric - OpenStack Heat
  - Container Centric – Kubernetes, Docker
A Lot of Relevant Tools Which I Won’t Cover Today :(

- **PaaS**: CloudFoundry, OpenShift, DEIS
- **Config Management**: Chef, Puppet, Ansible, Salt
- **Cloud Application Management**: Apache Brooklyn, Juju
- **Cloud management platforms**: RH CloudForms, Scalr
- **Cluster schedules**: Mesos, Fleet, Yarn, Nomad

This is by no means an exhaustive comparison!
The Test Application
1 Orchestration Process - Setup

Create network and compute resources: VMs, security group, network, subnet, routers, LB pool
Orchestration Process - Setup

Install Mongo and Node Binaries

- NodeJS
  - mongos
    - VM
- NodeJS
  - mongos
    - VM
- NodeJS
  - mongos
    - VM

Load Balancer

- Mongo-cfg
  - VM
- Mongo-cfg
  - VM
- Mongo-cfg
  - VM
3 Orchestration Process - Setup

Start mongod processes
Orchestration Process - Setup

Load Balancer

Start mongo-cfg processes

NodeJS
mongos
VM

NodeJS
mongos
VM

NodeJS
mongos
VM

Mongo-cfg
VM

Mongo-cfg
VM

Mongo-cfg
VM

mongod
VM

mongod
VM

mongod
VM
5 Orchestration Process - Setup

Start mongos processes, pointing to mongo-cfg servers
6 Orchestration Process - Setup

Pick one VM per shard and initialize replica set
Orchestration Process - Setup

Pick one mongos and add shards, one at a time.
Pick one mongos and initialize data in mongodb
9 Orchestration Process - Setup

Start nodejs processes

Load Balancer

NodeJS
mongos
VM

NodeJS
mongos
VM

NodeJS
mongos
VM

Mongo-cfg
VM

Mongo-cfg
VM

Mongo-cfg
VM

mongod
VM

mongod
VM

mongod
VM
Orchestration Process - Setup

Add nodejs VMs to LB pool

Load Balancer

NodeJS
mongos
VM

NodeJS
mongos
VM

NodeJS
mongos
VM

Mongo-cfg
VM

Mongo-cfg
VM

Mongo-cfg
VM

mongo
d VM

mongo
d VM

mongo
d VM
Beyond Just Setup

- Setup monitoring and log collection
- Manual/Auto healing
- Manual/Auto scaling
- "Day 2"
  - Backup and restore
  - Update code
  - Infrastructure upgrades and patches
Common Requirements

- Manage dependencies
- Reproducible
- Cloneable
- Recoverable
- Updateable
Category 1: “Pure-Play” Orchestration
Terraform by Hashicorp
Introduction to Terraform

◇ By Hashicorp
◇ Simple (in a good way) command line tool
  - Resources
  - Providers and provisioners
  - Modules
  - Variables and outputs
resource "openstack_compute_secgroup_v2" "nodejs_security_group" {
  name = "nodejs_security_group"
  description = "security group for mongodb"
  rule {
    from_port = 22
    to_port = 22
    ip_protocol = "tcp"
    cidr = "0.0.0.0/0"
  }
  rule {
    from_port = "${var.nodejs_port}"
    to_port = "${var.nodejs_port}"
    ip_protocol = "tcp"
    cidr = "0.0.0.0/0"
  }
}
# Create a Network

resource "openstack_networking_network_v2" "tf_network" {
    region = ""
    name = "tf_network"
    admin_state_up = "true"
}

# Create a subnet in our new network
# Notice here we use a TF variable for the name of our network above.

resource "openstack_networking_subnet_v2" "tf_net_sub1" {
    region = ""
    network_id = "${openstack_networking_network_v2.tf_network.id}"
    cidr = "192.168.1.0/24"
    ip_version = 4
}
Sample Configuration

resource "openstack_compute_instance_v2" "mongod_host" {
  count = "3"
  region = ""
  name = "mongod_host"
  image_name = "${var.image_name}"
  flavor_name = "${var.flavor_name}"
  key_pair = "tf-keypair-1"
  security_groups = ["mongo_security_group"]
  network {
    uuid = "${openstack_networking_network_v2.tf_network.id}";
  }
...
  provisioner "remote-exec" {
    scripts = [
      "scripts/install_mongo.sh"
      "start_mongod.sh"
    ]
  }
}
Solution Overview

◇ Single top level configuration file
◇ Creates: Network, subnet, router, floating IP, security groups, VMs, LBaaS pool
◇ TF module to model a mongodb shard
  ▪ No easy way to specify "I want X occurrences of this module"
  ▪ Just copy and paste...
Master Assignment & Registration of Shards

◇ Issue - no "cluster wide" way of invoking provisioners
  ▪ Needed for configuring shard masters and adding shards to the cluster
◇ Option 1: use Consul
  ▪ e.g. first instance acquires a lock and waits for other to join
◇ Option 2: Static allocation in the configuration
◇ Option 3: local-exec with locks
Kubernetes: Pros and Cons

Pros:
- Infrastructure & Framework neutrality
- Solid support for OpenStack
- Simple and elegant
- Present plan before applying
- Support for incremental updates

Cons:
- Configurations are not portable across cloud providers
- Hard to model non-infrastructure components
- Everything is done in the context of a single resource instance
What is TOSCA?

Topology Orchestration Spec for Cloud Applications

- **Topology**
  - Nodes, relationships, requirements & capabilities

- **Workflows**
  - Lifecycle operations implementation (install, scale...)

- **Policy**
  - Events, conditions, action mapping
What is TOSCA?
Cloudify: an Open Source Implementation of TOSCA

- Provision
- Configure
- Monitor
- Manage

Application Blueprint (TOSCA)

IaaS Plugins

Container Plugins

Conf Mgmt Plugins

Monitoring & Alarming
Cloudify: an Open Source Implementation of TOSCA

- Provision
- Configure
- Monitor
- Manage

- InfluxDB
- Riemann
- ElasticSearch

**Application Blueprint (TOSCA)**

**IaaS Plugins**
- openstack
- VMware
- Google Cloud Platform
- Amazon Web Services

**Container Plugins**
- Docker
- Kubernetes

**Conf Mgmt Plugins**
- Ansible
- SaltStack
- Fabric
- Chef
- Puppet Labs
Solution Overview

**Inputs:**
- # of nodeJS instances
- Mongo cluster details

**Outputs:**
- App Endpoint

**Inputs:**
- #config instances
- #Shards
- #Replica set per shard

**Outputs:**
- Mogoconfig hosts
- Shards endpoint
node_templates:
  nodecellar_security_group:
    type: cloudify.openstack.nodes.SecurityGroup
    properties:
      security_group:
        name: nodecellar_security_group
        rules:
          - remote_ip_prefix: 0.0.0.0/0
            port: {get_property:[ nodecellar, port ]}
Create Mongo Shards

mongodb:
  type: tosca.nodes.mongodb.Shard
directives: [substitutable]
properties:
  count: { get_input: number_of_replicas_per_shard }
requirements:
  - host:
    node: mongo_server
capabilities:
  scalable:
    properties:
      min_instances: 1
      max_instances: 10
      default_instances: { get_input: mongodb_rs_count }
mongo_server:
  type: tosca.nodes.Compute
  capabilities:
    host:
      properties: *host_capabilities
    os:
      properties: *os_capabilities
  scalable:
    properties:
      min_instances: 1
      max_instances: 10
      default_instances: 5
mongo_db_replica_set:
  type: tosca.nodes.DBMS
  requirements:
    - host:
      node: mongo_server
  interfaces:
    Standard:
      create: Scripts/mongodb/create.sh
      configure:
        implementation: Scripts/mongodb/config.sh
      inputs:
        mongodb_ip: { get_attribute: [mongo_server, addr] }
  start: Scripts/mongodb/start.sh

Create MongoDB Replica Set
Create NodeJS Containers

dodecellar_container:
  type: tosca.nodes.NodeCellarAppContainer
  properties:
    port: { get_input: nodejs_app_port }
  interfaces:
    cloudify.interfaces.lifecycle:
      create:
        inputs:
          ....
          command: nodejs server.js
          environment:
            NODECELLAR_PORT: { get_property: [SELF, port] }
            MONGO_PORT: { get_property: [SELF, database_connection, port] }
            MONGO_HOST: { get_attribute: [SELF, database_connection, private_address] }
Create Load Balancer

haproxy:
  type: tosca.nodes.Proxy
  properties:
    frontend_port: 80
    statistics_port: 9000
    backend_app_port: { get_property: [ nodecellar, port ] }
  requirements:
    - host:
      node: haproxy_frontend_host
    - member:
      node: nodecellar_container
Handling Post Deployment through Workflow & Policies

- **Built in workflows**
  - Install
  - Uninstall
  - Heal
  - Scale
  - Execute_operation

- Dependency awareness through graph navigation

- Remote/Local execution
TOSCA/Cloudify: Pros and Cons

Pros:
- Infrastructure & Framework neutrality
- Complete Life Cycle Management
- Handles Infrastructure & Software
- Post deployment handling*
  - Monitoring
  - Logging

Cons:
- The spec is still rapidly evolving, hard for vendors to keep up
- Cloudify isn’t 100% complaint yet
- Not enough implementations
- Not a lot of tooling

*Implementation specific
Category 2: Infrastructure Centric
Provides a mechanism for orchestrating OpenStack resources through the use of modular templates
Heat Architecture
Solution Overview

**Input:**
- # of nodeJS instances
- # of MongoDBConfig hosts
- # of MongoDB shards hosts

**Input:**
- # of config instances

**Input:**
- # of replicas per shard

**Output:**
- mongos node hosts
- App EndPoint = LoadBalancer IP/path

**Output:**
- mogocfg node hosts

**Output:**
- Replica set node hosts
  ssh-key, private ip to the init node
Infrastructure setup

resources:
secgroup:
  type: OS::Neutron::SecurityGroup
  properties:
    name:
      str_replace:
        template: mongodb-$stackstr-secgroup
        params:
          $stackstr:
            get_attr:
              - stack-string
              - value
    rules:
      - protocol: icmp
      - protocol: tcp
        port_range_min: 22
        port_range_max: 22
      - protocol: tcp
        port_range_min: 27017
        port_range_max: 27019
mongo_host:
  type: OS::Nova::Server
  properties:
    name:
      str_replace:
        template: $stackprefix-$stackstr
        params:
          $stackprefix:
            get_param: stack-prefix
          $stackstr:
            get_attr:
              - stack-string
              - value
    image:
      get_param: image
    flavor:
      get_param: flavor
    security_groups:
      - get_param: security_group
mongodb_peer_servers:
  type: "OS::Heat::ResourceGroup"
properties:
  count: { get_param: peer_server_count }
resource_def:
  type: { get_param: child_template }
properties:
  server_hostname:
    str_replace:
      template: '%name%-0%index%'
      params:
        '%name%': { get_param: server_hostname }
  image: { get_param: image }
  flavor: { get_param: flavor }
  ssh_key: { get_resource: ssh_key }
  ssh_private_key: { get_attr: [ssh_key, private_key] }
  kitchen: { get_param: kitchen }
  chef_version: { get_param: chef_version }

Create MongoDB Replica Servers
server_setup:
    type: "OS::Heat::ChefSolo"
depends_on:
    - mongodb_peer_servers
properties:
    username: root
    private_key: { get_attr: [ssh_key, private_key] }
    host: { get_attr: [mongodb_peer_servers, accessIPv4, 0] }
    kitchen: { get_param: kitchen }
    chef_version: { get_param: chef_version }
node:
    mongodb:
        ruby_gems:
            mongo: '1.12.0'
            bson_ext: '1.12.0'
        bind_ip: { get_attr: [mongodb_peer_servers, privateIPv4, 0] }
        use_fqdn: false
        replset_members: { get_attr: [mongodb_peer_servers, privateIPv4] }
        config:
            replset: myreplset
        run_list: [ "recipe[config_replset]" ]
nodestack_chef_run:
  type: 'OS::Heat::ChefSolo'
  depends_on: nodestack_node
  properties:
    ...
    node:
      nodejs_app:
        ...
      deployment:
        id: { get_param: stack_id }
        app_id: nodejs
        run_list:
          "recipe[apt]",
          "recipe[nodejs]",
          "recipe[ssh_known_hosts]",
          "recipe[nodejs_app]"
  data_bags:
    nodejs:
      id: { get_param: stack_id }
    nodejs_app:
      password: { get_attr: [nodejs_user_password, value] }
      deploy_key: { get_param: deploy_key }
      database_url:
        str_replace:
          template: 'mongodb://%dbuser%:%dbpasswd%@%dbhostname%'
          params:
            '%dbuser%': { get_param: database_username }
            '%dbpasswd%': { get_param: database_user_password }
            '%dbhostname%': { get_param: db_server_ip }
Heat: Pros and Cons

Pros:
 ◇ Native To OpenStack
 ◇ Built-in mapping of most OpenStack infrastructure resource types

Cons:
 ◇ OpenStack Only
 ◇ Software configuration is limited
 ◇ Lack of built-in workflow abstraction
 ◇ Post deployment orchestration is limited
   ▪ Requires integration with other tools/projects
Category 3: Container Centric
Docker Swarm
Docker Swarm

- A Docker-native clustering system
- Use a pool of hosts through a single swarm master endpoint
- Augmented by Compose and Machine
- Placement constraints, affinity/anti-affinity
- Native docker experience

```
docker run -name rs1 -e affinity:container!=rs*
```
Swarm Architecture
for i in 1..{number_of_replica_sets}
  for j in 1..{number_of_nodes_for_replica_set}
    docker run \
      -name rs{i}_srv{j} \n      -e affinity:container!=rs* \n      -e affinity:container!=cfg* \n      -e constraint:daemon==mongodb \n      -d example/mongodb \n      --replSet rs{i}

Then, SSH into one host per replica set to configure it.
Make sure you inject all mongos endpoints for the application.

```bash
for i in 1..{number_of_nodejs_servers}
docker run -P -name nodejs{i}_v1 -e constraint:daemon==nodejs -e affinity:container!=nodejs* -e MONGO_HOSTS=<LIST_OF_MONGOS_IPs> -d example/nodejs_v1 nodejs server.js
```
Extract Node.js container IPs using docker inspect and then:

```bash
for i in 1..{number_of_nodejs_servers}
docker exec haproxy1 \
reconfigure.sh \
--add=<IP_of_nodejs{i}:port>
```
Solution Overview

MongoDB Scale Out

Identical to the process of deploying the initial MongoDB shards, MongoDB will take care of migrating data to the new shard.
Docker Swarm: Pros and Cons

Pros:
- Easy modeling
- Placement/Affinity
- Simplicity
- Docker native experience
- Scalability

Cons:
- Container only
- No infrastructure handling (Docker Machine can help)
- Scripting to do anything more than place & start containers
- No topology awareness (Docker Compose can help)
- Requires other tools for post deployment aspects - monitoring, healing, scaling
Kubernetes
Quick Intro to K8s

Container deployment management

- **Pods**: a tightly coupled group of containers
- **Replication controller**: ensures that a specified number of pod "replicas" are running at any one time.
- **Networking**: Each pod gets its own IP address that's visible to other pods
- **Service**: Load balanced endpoint for a set of pods
K8s Architecture

Master components Collocated, or spread across machines, as dictated by cluster size.
apiVersion: v1beta3
kind: ReplicationController
spec:
  replicas: 5
  selector:
    name: mongod-rs1
template:
  metadata:
    labels:
      name: mongod-rs1
  spec:
    containers:
    - command: [mongod, --port, 27017, --replSet, rs1]
      image: example/mongod
      name: mongod-rs1
    - command: [mongod-rs-manager, --replSet, rs1]
      image: example/mongod-rs-manager
      name: mongod-rs1-manager
apiVersion: v1beta3
type: Service
name: nodejs
spec:
  createExternalLoadBalancer: true
  selector:
    type: nodejs
  ports:
    - port: 80
targetPort: 8080
Solution Overview – Deploy

◇ Create mongod config servers

```bash
for i in 1..3
    kubectl create -f mongod-configsvr{i}-controller.yaml
    kubectl create -f mongod-configsvr{i}-service.yaml
```

◇ Create mongos router

```bash
kubectl create -f mongos-controller.yaml
kubectl create -f mongos-service.yaml
```
for i in 1..{number_of_replica_sets}
  kubectl create -f \\
      mongod-rs{i}-controller.yaml

  # Now configure each replicate set
  # by picking pod to be the initial “master”
  # of each replica set and extract all
  # containers IPs using “kubectl get -l ...

  # dynamically update replica set
  # members (this will kick off this process)
  kubectl create -f mongod-rs{i}-service.yaml
Failing pods are identified by Kubernetes and are automatically rescheduled.
Kubernetes: Pros and Cons

Pros:
- (almost) zero configuration autoheal
- Out of the box load balancer
- Simple scaling

Cons:
- Container only
- No placement (yet)
- Not simple to manage stateful services
- No topology awareness
So what’s the Right Tool for You?

◇ Are you hell bent on containers? OpenStack?
◇ Do you have legacy workloads?
◇ Do you consider infrastructure resources as part of the process?
◇ Do you have a heterogenous environment?
◇ Do you want the same tool to also handle post deployment?
The Tools Presented Are Not Necessarily Mutually Exclusive!
Final Words
"It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is most adaptable to change."

Charles Darwin
Thank You

Questions?
Find us at:
◇ Twitter @CloudifySource
◇ email info@getcloudify.org