MiCADO infrastructure set-up

prerequisite:

CloudBroker account: https://cloudsme-prototype.cloudbroker.com/

Introduction:

This document will help the administrator to set up his own MiCADO infrastructure. Currently the version of the infrastructure is 0.03 and it's not currently for Production use, but it's a great tool for development use.

Setup:

- **First step: update of the necessary config files**

A setup.sh script has been created to update the config.json file, create a ssh key-pair for the manager and put the admin public key inside the authorized_keys file of the manager. Before running it you should have the Consul key. The command line to run it is: ***sh setup.sh***

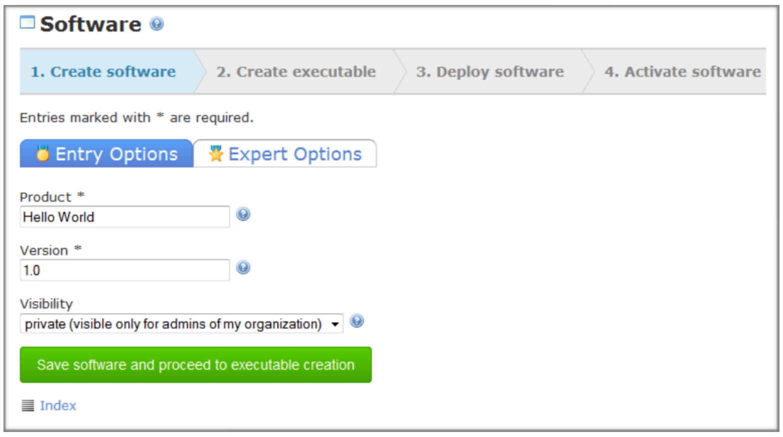
You can produce the consul encrypted key by using the consul binary using it as follow: ***consul keygen***
You can get the consul binary here: https://releases.hashicorp.com/consul/0.7.3/consul_0.7.3_linux_amd64.zip

The config.json is as follow:

```
{
     “server":true,
     "bootstrap_expect":2, “data_dir”:”/var/consul",
     "node_name":"manager",
     “encrypt”:”consul encrypted key",
     "bind_addr":"31.171.246.150",
     "datacenter":"DSCR", "log_level":"err",
     "addresses":{
            "http":"0.0.0.0"
     },
      “enable_syslog":true
}
```

Create each packaged (ie manager/node/proxy) using the format tar.gz calling it input.tar.gz and putting everything apart from the deploy.sh script that will be used to installed the instances on CloudBroker.

- **Second step: creation of a new softwares in CloudBroker**



Entries marked with * are required.

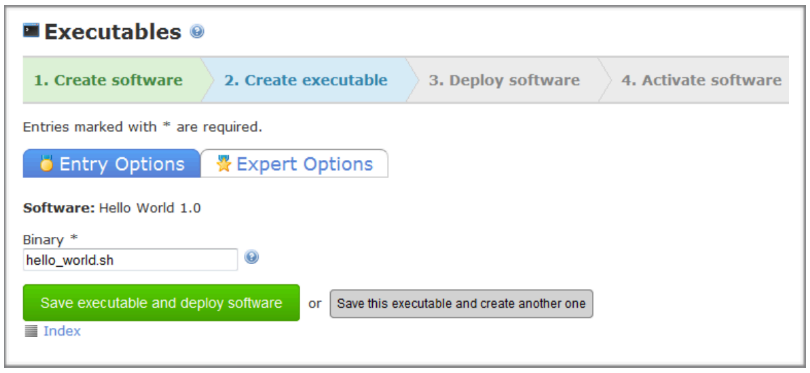
Product *
Hello World

Version *
1.0

Visibility
private (visible only for admins of my organization)

Clic on new software and fill the field with the name you want to give and the version.

Then clic on “Save software and proceed to executable creation”



Entries marked with * are required.

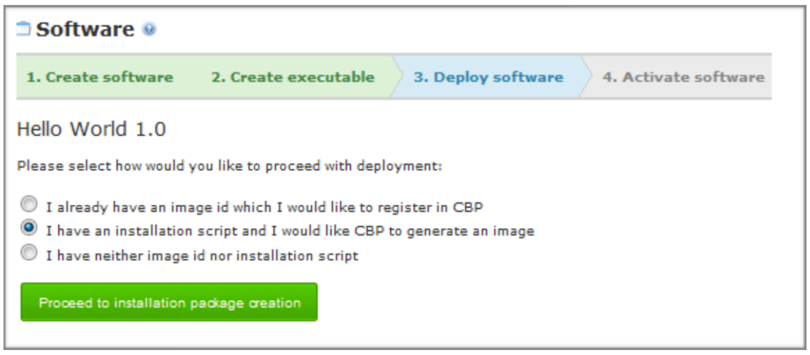
Software: Hello World 1.0

Binary *
hello_world.sh

Fill the Binary field with the executable that will be executed when the job is launch. (it should be name executable.sh)

Then clic on “Save executable and deploy software”

If a port needs to be open to access the instance (ie 22) add it in the Expert Options. (for manager open port 22 or 2222 if on CloudSigma, open nothing for node, and open port 80 for the proxy)



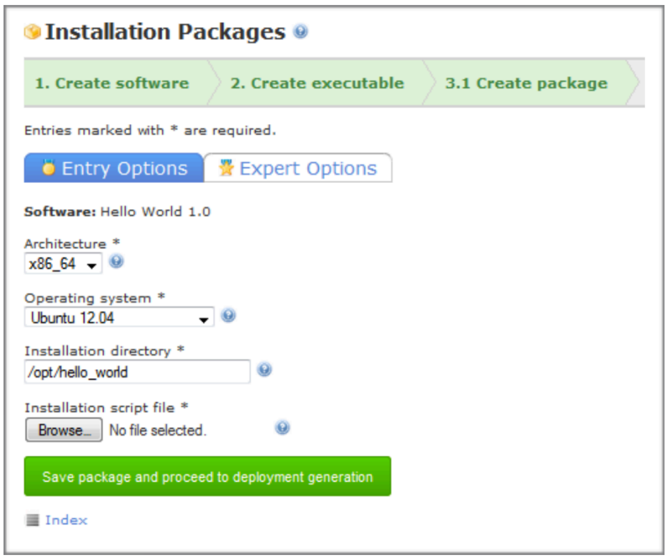
Software

Hello World 1.0

Please select how would you like to proceed with deployment:

- ○ I already have an image id which I would like to register in CBP
- ● I have an installation script and I would like CBP to generate an image
- ○ I have neither image id nor installation script

Choose the option “I have an installation script and I would like CloudBroker Platform to generate the image”



Installation Packages

Entries marked with * are required.

Software: Hello World 1.0

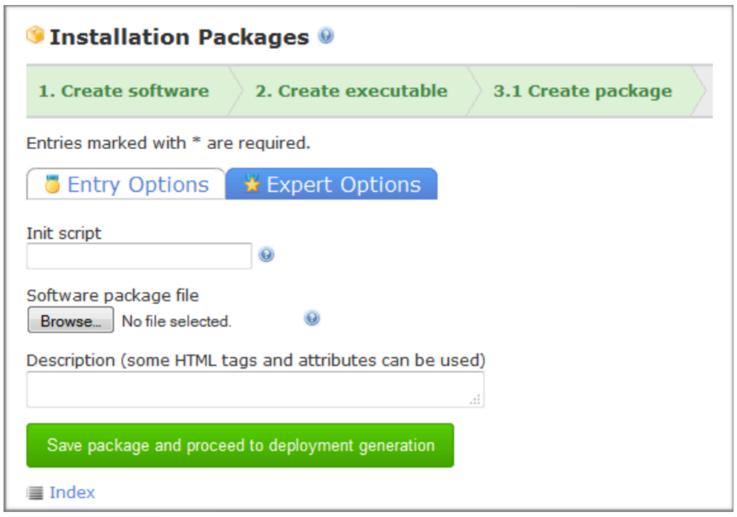
Architecture *: x86_64

Operating system *: Ubuntu 12.04

Installation directory *: /opt/hello_world

Installation script file *: Browse... No file selected.

Choose the image wanted by selecting the different option with the drop down menu. (best to use x86_64 ubuntu 14.04). The installation directory should be (/opt/ DSCR_manager or /opt/DSCR_node or /opt/ proxy depending on which software you are creating)



Installation Packages

1. Create software | 2. Create executable | 3.1 Create package

Entries marked with * are required.

Entry Options | Expert Options

Init script

Software package file

Browse... No file selected.

Description (some HTML tags and attributes can be used)

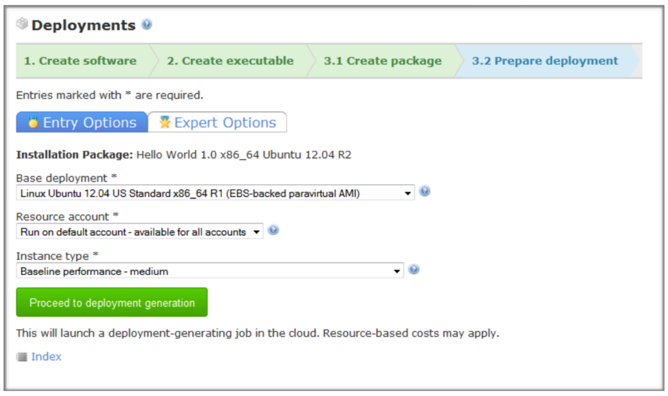
Save package and proceed to deployment generation

Index

Upload the installation script which is deploy.sh.

Go to Expert Options and upload the install.tar.gz file.

Once the files uploaded clic on “Save package and proceed to deployment generation”.



Deployments

1. Create software | 2. Create executable | 3.1 Create package | 3.2 Prepare deployment

Entries marked with * are required.

Entry Options | Expert Options

Installation Package: Hello World 1.0 x86_64 Ubuntu 12.04 R2

Base deployment *
Linux Ubuntu 12.04 US Standard x86_64 R1 (EBS-backed paravirtual AMI)

Resource account *
Run on default account - available for all accounts

Instance type *
Baseline performance - medium

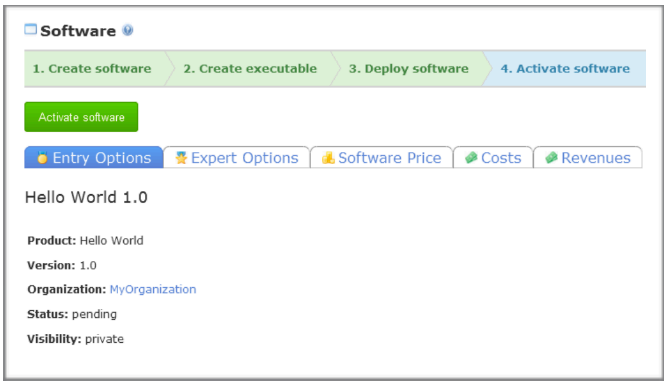
Proceed to deployment generation

This will launch a deployment-generating job in the cloud. Resource-based costs may apply.

Index

Choose the base-deployment (best to use linux from CloudSigma) take the smallest flavour (an image can always scale up but not down).

Clic on “Proceed to deployment generation”.



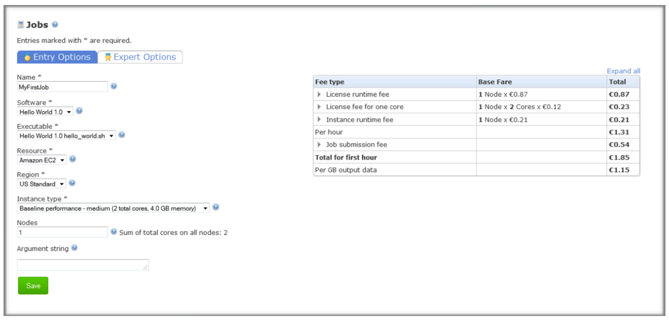
Hello World 1.0

Product: Hello World
Version: 1.0
Organization: MyOrganization
Status: pending
Visibility: private

We can see the job running. When the job is done clic on "Activate software".

First use:

Launch a permanently running instance of the manager.



Name *: MyFirstJob
Software *: Hello World 1.0
Executable *: Hello World 1.0 hello_world.sh
Resource *: Amazon EC2
Region *: US Standard
Instance type *: Baseline performance - medium (2 total cores, 4.0 GB memory)
Nodes: 1 — Sum of total cores on all nodes: 2
Argument string:

Fee type	Base Fare	Total
License runtime fee	1 Node x €0.87	€0.87
License fee for one core	1 Node x 2 Cores x €0.12	€0.23
Instance runtime fee	1 Node x €0.21	€0.21
Per hour		€1.31
Job submission fee		€0.54
Total for first hour		€1.85
Per GB output data		€1.15

Clic on add job, in the job tab. This page will appear, then choose the software to launch i.e. the manager. Go to the expert option and check box for ***permanent running instance*** and check ***I still want this job to run an executable and wait for further jobs after.***

Then clic on Actions -> Submit.

Once the job is running then you can ssh to the manager (if on CloudSigma, use the port 2222).

Finish the set up to be able to launch the instances through the manager instance via CloudBroker API, modify cbcredentials file inside /opt/manager_DSCR/input directory by changing it to your credentials, and you're all set up. Also edit the /opt/manager_DSCR/cloudsetup.xml file to add the information on the cloud and on the application to launch.

Command Line to use the Infrastructure to launch jobs

Those command line are available in the main.py script located in /opt/manager_DSCR/autoscaling however a soft link as been created called micado that would run this script as described bellow.

different command:

start:

This command use the ClouldBroker API to launch instance to be added to the infrastructure.

stop:

This command stop the desire instance running in the infrastructure.

launch:

This command launch a container on the infrastructure.

quit:

This command allow to stop a running container.

list:

This command list either the container or the instances running on the infrastructure.

build:

This command allow to build a new docker image.

info:

This command gives info on the running containers (special for the developers)

logs:

This command give back the logs of the wanted container to be able to debug them.

remove:

this command allow to remove the wanted container (the container should not be on running mode).

execute:

this command allow to execute any command inside the container.

different parameter for each commands:

start:

-c / --cloud: allow to choose on which Cloud to start a new instance. -a / --application: allow to choose which instance to run.
-f / --flavour: allow to choose which flavour for the instance.
-t / --tag: assign a Tag to the instance to launch.
-n / --name: choose on either application or a database server.
-l: parameter to use for launching the first instance.

example

micado start -c CloudSigma -a “Node DSCR” -f Small -t application01 -n application

stop:
-t / --tag: the name of the tag given for the instance we want to stop.

example
micado stop -t application01

launch:
-i / --image: the name of the image container wanted.
-n / --name: the name of the client to the container is launched for.
-l / --location: where the container should run.
-s / --script: inside path of the script to be run at the execution of the container. -d / --domain: domain name to access the application run in the container run. -a / --arguments: extra argument to pass to the container when it will be run. -p / --port: choose to run the container on a specific port of the instance.
-e / --environment: allow to set an Environment variable inside the container.

example
micado launch -i container-default -n client -l application -s /usr/src/app/app.sh -d domain -a user host_ip application address port

quit:
-n / --name: name of the container to stop
-rm: option to remove the container after it has been stopped.

example
micado quit -n container-abc -rm

list:
-i / --instance: list all the instance up in the infrastructure.
-c / --containers: list all the containers running in the infrastructure.
-a / --all: optional argument for the listing all the container stop or not in the infrastructure.

example
micado list -c -a

build:
-n / --name: name to give the image of the docker image. This name should have the pattern of “container-name”
-f / --filename: tarball containing the Dockerfile and all other files needed. those file should be inside a directory having the name you want to give to the container.

example
micado build -n container-application -f tarball.tar

info:
-s / --service: name of the services to get the info from -a / --all: list all the service running on the infrastructure

example
micado info -a

logs:
-n / --name: container name to get the logs from.

example
micado logs -n container-x

remove
-n / --name: name of the container to remove, it should not be running.

example
micado remove -n container-x

execute
-n / --name: name of the container to execute the command in. -c / --command: command to be executed inside the container.

example
micado execute -n container-x -c bash

current state of the infrastructure

What composes the current state infrastructure: docker, swarm, consul, registrator. In this version of the infrastructure, we have the starting investigation of the scalability and is still in its first steps.

definition of the tool of the infrastructure:

- Docker: Light-weight virtualisation built on linux containers.

- Swarm: Framework to create a cluster of nodes running Docker, distribute and load-balance docker containers. When a new container is launch on the infrastructure the swarm service will check on which node to run it. Some constraint can be added, for the user to decide on which kind of nodes he wants the container to run.

- Consul: Service discovery tool to manage Swarm cluster, but also services provided by containers in the cluster.

- Registrator: Tool to register running containers with Consul.

- Mongodb: database used for this version of the infrastructure for the scalability. In this database there is three tables: container, instance and imagecontainer. For the container table we will have details on the memory usage, the name, the ip address etc. For the instance table we will have details on the memory left, the name of the instance, the ip address etc. For the imagecontainer we will have information on the name of the image and it's critical usage. A cron job as been set up to update every few minutes the informations on the infrastructure

Docker:

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

Swarm:

On this version of the infrastructure, swarm is now set up directly using the swarm container. This is setup by the init.sh script that is launched by the execute.sh script when the instance is launched, the execute script will be run. The init script will launch the swarm container on the manager node. For the swarm container to run a token needs to be created by running a docker command, that will put the token inside a file that will be send to every new instance that will be spawned to the infrastructure for its swarm to join the cluster.

Consul:

For the consul to be able to discover services, The consul needs to have the configuration file (config.json). This file contain the consul encrypt key. It's in this file too that the name of the node is given (all the instance belonging to the infrastructure needs to have a different name), and it need to have the IP address to be written in it to complete the configuration. The consul.conf file will allow to start the service once the instance is fired up. When the manager instance is fired up, and that everything is running, the IP address of the manager will be put inside a file calle manager_ip.txt for it to be passed to the nodes launched afterwards. The nodes will use this manager_ip.txt to join the consul cluster as it will know which one to join.

Registrator:

The Registrator is a container that is launched only on the node of the infrastructure, not the manager. It is launched through the execute script which is executed when a node instance is fired up.

Private Registy:

The private registry is used to store container images. Thanks to the private registry, the node is able to pull the wanted container to run on it. Without the private registry, each image container needs to be installed on every nodes of the infrastructure, if the node doesn’t have the wanted container image then Swarm cannot decide to run the container on it even though it’s the more appropriate node to be run on.

Composition of files for manager node

deploy.sh:
this script will be use to create the correct image for the application.

execute.sh:
script that is executed when the instance is fired up.

Cloudbroker.py:
author Hannu Visti this python script allow the communication between CloudBroker through the API and the infrastructure.

cloudsetup.xml:
author Hannu Visti used by the Cloudbroker.py python to select the wanted image and cloud when launching a new instance.

consul.conf:
configuration file for consul to allow the service to be up and running when the instance is fired up.

config.json:
configuration file for consul to name the node, give the consul encryption key
init.sh:
initialisation script, that will update the configuration script with IP address of the instance, and restart all the services needed after the update. it will launch as well the swarm service through docker.

main.py:
author Gregoire Gesmier use to start new instance on the wanted cloud with the wanted flavour. Stop instance if needed, listing the instance running, container running, and the image of the container. container_quit to stop the running container. container_build to build the container, launch_container to launch the container, get_info to get the info of the container. container_logs, container_remove, container_execute, copy_file. In this version when we will launch a container, then a check is made to see if there's on the infrastructure an instance able to have the container run on it. If not then a new instance will be launched, the script will wait for the instance to be up and running and then proceed to the launch of the container.

Composition of files for node

config.json:
configuration file for consul to name the node, give the consul encryption key, for the node to be able to be discover by the infrastructure.

consul.conf:
configuration file for consul to allow the service to be up and running when the instance is fired up.

deploy.sh:
this script will be use to create the correct image for the application.

docker-config.json:
config file to be able to connect to the private registry. This file will contain the password to be able to connect to the private registry.

execute.sh:
initialisation script, that will update the configuration script with IP address of the instance, and restart all the services needed after the update. it will launch as well the swarm service through docker.