HelloWorld Service in XOS

Sapan Bhatia
Open Networking Lab

October 30, 2015

Introduction

This document walks through the implementation of a simple HelloWorld service on the XOS platform. It provides a technical overview of the service and its components, as well as instructions for setting it up.

The HelloWorld service displays the concatenation of two strings in a web page, where the user (tenant) of the service specifies the two strings. The service then sets up a new Instance (VM), and configures it to install the Apache web server at the time of its first start up. The server then displays the two strings specified by the user of the service in a web page.

The current version of HelloWorld is minimal. It includes a single instance for the whole service and it does not synchronize changes to the strings. We plan to expand the example over time to illustrate additional features.

Setting Up the Service

The following assumes a working XOS installation. See the *Configuration Management* section of the *Developer Guide* (http://guide.xosproject.org/2_developer/) for instructions on installing XOS. The “Devel Config” is sufficient for setting up and running the HelloWorld service.

To set up HelloWorld, follow the steps below. Unless otherwise specified, file and directory paths are relative to the XOS installation directory: */opt/xos*.

1. Register HelloWorld as an add-on component by adding the string “helloworld” to the list INSTALLED_APPS in the file *xos/settings.py*. This variable should contain the list of preconfigured services in XOS.

    'services.onos',
    'ceilometer',
    'helloworld',
    'requestrouter',
    'syndicate_storage'

2. Add the service’s models to the database using Django’s automatic migration feature:

    python manage.py makemigrations
    python manage.py migrate

3. Generate a dependency map for the service. The dependency map links objects in the HelloWorld service to objects in other services, such as the VM management core service.

    ./dmdot helloworld > helloworld-deps
    cp helloworld-deps model-deps

4. Restart the xos platform services:

    supervisorctl restart observer
    scripts/opencloud stopserver

    At this point, your docker container should have exited, in which case, you would need to restart it. Note the container id in the output of the first command, and then run it:

    docker ps -a
    docker run *<container id>*

Using the Service

To deploy the HelloWorld service, first set up a Slice, and create an Instance within that slice. That instance will be copied into a new one to host the HelloWorld service.

To deploy a new instance of the HelloWorld service, go to the XOS portal at URL:

http://*<xos-head-node>*/helloworld

Next, configure the two strings with values of your choice. These strings are not interpreted. They are displayed in the output of the service as is. When you have entered the submitted the form, you should see a new instance appear within your slice. The Instance should automatically be set up with the HelloWorld service, namely the Apache web service configured to display the strings you entered in the Service creation form.

To test the service, go to the following address in your web browser:

http://<*ip address of VM*>/hello.txt

You can find out the IP address of your VM by looking in the list of Instances in your slice. If your instances are configured with private IP addresses (as is the case on CloudLab), consider using the lynx web browser to see the output of the service:

```
apt-get -y install lynx
lynx http://<ip address of VM>/hello.txt
```

Components

The HelloWorld service has three main components: a set of one or more *Models* that define the authoritative state for the service, a *Synchronizer* that keeps the back-end system that implements the service in sync with this state, and a *View* that provides an interface to the service. The models, which are used to extend the XOS data model, can be found in the file *helloworld/models.py*; the synchronizer is under *synchronizers/helloworld*, and the view is in *helloworld/views.py.* The following discusses each of the three components in more detail.

Models

The XOS data model consists of a set of core models (e.g., instances, networks, slices, services, deployments) and a collection of service-specific models (e.g., HyperCache, ONOS, Syndicate, Ceilometer). These latter models hold a collection of configuration values and control parameters for the corresponding service. In the case of the HelloWorld service, there are two objects (Django models): Hello and World. Both of these objects contain a string in the field “name.” The HelloWorld service displays the concatenation of these two strings in a web page.

These two objects also contain links to other objects. The World object contains a link to the Hello object, through the “hello” field. This link is an illustration of an intra-service dependency. Through the dependency map generated in Step 3 of the setup, the

operation of the service is constrained so that dependent Hello objects are made operational (“synchronized”) before World objects.

The Hello object contains a link to an Instance object, which represents a provisioned virtual machine. The name of the field, “instance_backref”, contains the suffix “_backref” to indicate a reversal of the polarity of the dependency. Instead of defining a link from Hello to Instance, it makes Instance objects depend on Hello objects. The outcome of this dependency is to guarantee that Hello objects are synchronized before their accompanying Instance objects. This ordering is a requirement for the Hello World service, since Instances are preconfigured with the packages they need to contain, and the content that they are to serve.

Synchronizer

The Synchronizer makes the service operational. It does this by converting the configuration specified in the data models into directives for the back-end platform. This corresponds to bringing up VMs, configuring those VMs, and issuing commands to the OS.

For each object (Django model), the synchronizer contains a synchronization step. There are two parts to each step: a Python wrapper that acts a as a translator between the service’s abstract configuration and the concrete configuration of the back-end system, and an Ansible playbook that actually communicates the latter configuration to the back-end. More details on the anatomy of the synchronizer and its steps can be found in the *Developer Guide*. The following looks at the HelloWorld synchronizer in more detail.

The goal of the HelloWorld synchronizer is to instantiate a new VM with an Apache Web server installed, and drop in a web page containing text provided by the user. Since it relies on the the core VM management service in the *openstack_synchronizer/* directory to instantiate Instances, the HelloWorld synchronizer does not contain any Ansible modules itself. Instead, it feeds values to the Instance synchronizer. Specifically, it adds instructions to “user_data”, a set of configuration directives to the “cloud-init” initialization daemon.

```
nova_compute:
    auth_url: {{ endpoint }}
    login_username: {{ admin_user }}
    login_password: {{ admin_password }}
    login_tenant_name: {{ admin_tenant }}
    name: {{ name }}
```

```
{% if delete -%}
state: absent
{% else -%}
state: present
availability_zone: {{ availability_zone }}
image_name: {{ image_name }}
wait_for: 200
flavor_name: {{ flavor_name }}
user_data: "{{ user_data }}"
config_drive: yes
```

The instructions for installing the Apache web server and adding a web page to be served are implemented as follows.

```
sync_hello.py:

instance.userData="packages:\n  - apache2\nruncmd:\n  - update-rc.d apache2
enable\n  - service apache2 start\nwrite_files:\n-   content: Hello %s\n    path:
/var/www/html/hello.txt"%record.name
instance.save()
```

View

The user-facing interface to configuring a service is called a view. It takes inputs from a user and creates the appropriate XOS objects to represent the desired behavior. The View is written in Python using Django, and intercepts and responds to HTTP requests. The View for the helloworld service is implemented in the file *helloworld/views.py*.

Note that this example view is minimal. A more complete description of how to construct a view can be found in the *Adding Views to XOS* section of the *Developer Guide* (http://guide.xosproject.org/2_developer/).

A new mechanism to auto-generate views is under development. This section will be updated to use this new mechanism once it is ready.