Chapter 15
Operation Control Interfaces

Craig Sheridan and Darren Whigham

15.1 Introduction

An interesting commercial use-case for Flexiant of the MODAClouds solution is based upon adding extra functionality to Flexiant Cloud Orchestrator (FCO) [1] Triggers [2]. Triggers are functions that allow an action in FCO to initiate a second action, which can either be internal or even external to Flexiant Cloud Orchestrator.

A trigger is simply a block of Flexiant Development Language (FDL) [3] code, which is a Lua [4] based language, that is used to extend Flexiant Cloud Orchestrator.Triggers will run either before an event occurs (a pre trigger) or after an event occurs (a post trigger) which can be used to perform a variety of actions such as automatically starting servers at creation time or mailing and alerting based on customer actions.

15.2 Language for Triggers Description

A trigger is written as a block of Flexiant Development Language (FDL) code, which is a Lua based language used to extend Flexiant Cloud Orchestrator. FDL is written as a code block and run within the platform itself. Within FDL there are multiple APIs [5] that can be called such as billing, trigger, and payment. For this chapter we will focus on the trigger API. Triggers can be used to perform any of the following actions:

C. Sheridan · D. Whigham (✉)
Flexiant, London, UK
e-mail: dwhigham@flexiant.com

C. Sheridan
e-mail: csheridan@flexiant.com

E. Di Nitto et al. (eds.), *Model-Driven Development and Operation of Multi-Cloud Applications*, PoliMI SpringerBriefs,
DOI 10.1007/978-3-319-46031-4_15

- Sending email as a result of an action or state change
- Making an HTTP call and processing the result
- Making user API or admin API calls from within FCO.
- Writing an entry in the syslog
- Running a local executable file
- Reading or writing to a file
- Manipulating XML documents, objects, and nodes

To achieve these different actions, various trigger types are utilised. The following table lists these different 'triggerTypes', as well as whether the trigger is initiated before (PRE) or after (POST) the initiating event.

15.3 Architecture of the Trigger Support

The FDL Trigger API, named "TRIGGER", is activated when the user returns from an entry point with the API set to TRIGGER. This makes the entry point a trigger. Once an entry point has been set to act as a trigger, it will return three pieces of additional data when describing themselves; triggerType, triggerOptions, and value object.

The triggerType is the type of event that will initiate the trigger, for example an API call or a change in resource state. This can be refined using the triggerOption object, which is a list stating the specific events that can initiate the trigger. For example, if the triggerType indicates that the trigger can be initiated by a server state change, the triggerOptions determine which server states initiate the trigger.

The value object has the same layout as in the API (See SOAP Value). Each value object specifies a configurable value, together with its validator, thus setting out the permissible values for it.

With all FDL APIs, including TRIGGER, anything which a user prints (to STD-OUT or STDERR) will go to the Jade sysout log. The user can log any string to the normal log with logger (which takes a string). If the Lua throws an exception, Jade will catch it. However, the user should aim not to throw exceptions but instead return something appropriate depending on the API.

The entry point will always be called with a single parameter *p* dependent on the API being called, or a value of nil. If a value other than nil is passed, the return value of the function depends upon the API. In this case, the function is expected to return a table that describes itself. This table will contain the following keys:

- *api*: the name of the API as a string (for instance "BILLING")
- *version*: the version of the API as a number.
- *ref*: a unique identifier for the function. Do not use identifiers starting with an underscore; these are reserved for Flexiant.
- *name*: a string containing the name of the entry point (max 50 characters)
- *description*: a string containing a description of the entry point (maximum 250 characters)

- *execution function*: a reference to a LUA function which is the function to call with values of p other than nil. If this is not specified or is specified as nil, then the same function will be called.

15.4 Usage of Triggers to Enable Load Balancing

Triggers are most commonly used to access all the functionality that is offered by FCO, but they can also be used to make external API calls. Trigger functionality has been added as part of the MODAClouds project to extend the platform and tools capabilities. Within the MODAClouds project a number of unique triggers have been developed.

The first of these triggers is called the Auto Server Failover trigger, which is called should a server be shutdown or killed within a certain customers account.

Upon being called this trigger looks for a Live Server tag attached to the server, and if found, replaces it with a Backup server tag. This new tag can be anything, such as Faulty Server, but for this example Backup server will be used. The trigger then looks in the FCO account for a VM tagged Backup server that is in a stopped state and starts it. Finally, once the new server is started, the Backup server tag is removed and a Live server tag is added.

Another trigger that has been created for the MODAClouds project is the Auto Alert Mail trigger. This will send an email to the account owner to alert them that a server has stopped or been killed. The Auto Alert Mail trigger works by looking for an "Auto Mail" tag assigned to the relevant account whenever FCO registers that a server has been shutdown or killed. This tag contains the recipient address to send an email to, and once found, the trigger sends a message to the address to inform the account holder that a server has been shut down. The message includes the UUID [6] of the server and the Date/time stamp for when this server was shutdown. This useful trigger therefore allows account owners to be notified of any issues with their servers, as well as recording a date/timestamp within the syslog to allow for troubleshooting.

Both of the Auto Server Failover and Auto Mail Alert triggers have been combined and included within the MODAclouds solution as detailed in the following section.

Within the Modacloud project, these triggers have been implemented to work in conjunction with load balancers. As detailed in Fig. 15.1.

The Load Balancer will be set up within the FCO Cloud platform. Behind this will be a number of VM's that will serve load balancers. These VMs will be tagged within FCO as either a Live Server or a Backup Server In the event of an error with these servers that cannot be resolved internally, the server is then shutdown. When this shutdown occurs then the triggers created.

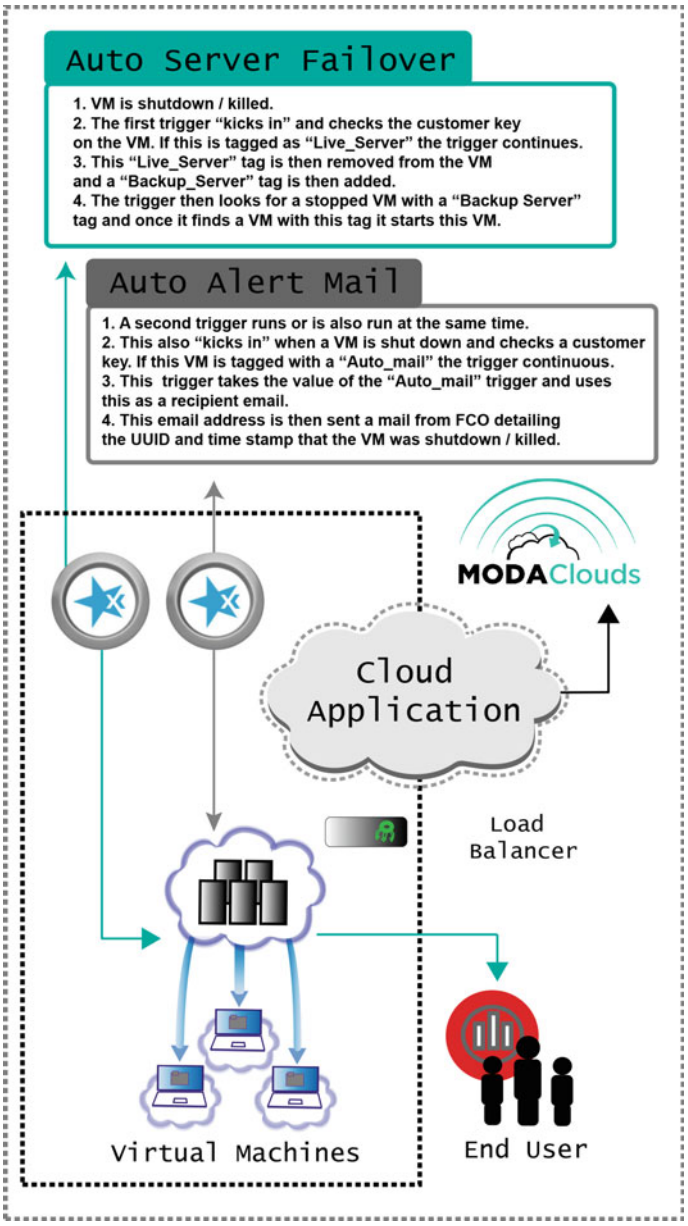
Auto Server Failover
1. VM is shutdown / killed.
2. The first trigger "kicks in" and checks the customer key on the VM. If this is tagged as "Live_Server" the trigger continues.
3. This "Live_Server" tag is then removed from the VM and a "Backup_Server" tag is then added.
4. The trigger then looks for a stopped VM with a "Backup Server" tag and once it finds a VM with this tag it starts this VM.
Auto Alert Mail
1. A second trigger runs or is also run at the same time.
2. This also "kicks in" when a VM is shut down and checks a customer key. If this VM is tagged with a "Auto_mail" the trigger continuous.
3. This trigger takes the value of the "Auto_mail" trigger and uses this as a recipient email.
4. This email address is then sent a mail from FCO detailing the UUID and time stamp that the VM was shutdown / killed.
MODAClouds
Cloud Application
Load Balancer
Virtual Machines
End User

Fig. 15.1 MODAcloud triggers

15.5 Related Work

To be able to monitor and provide similar solutions that are presented here with other Cloud providers external tools/programs using the Cloud providers APIs must be used. To be able to match this functionality providers such as OnApp, VMWare and OpenStack would have to look at using external API calls.

Within FCO and with the use of Triggers and FDL, FCO allows the ability to run and monitor from within the platform rather than using external applications to query using the API. The key benefit of this from a Cloud provider is the reduction in the number of external API calls and the functionality works regardless of the hypervisor/storage/network model underneath.

15.6 Conclusions

This chapter has provided an overview of the trigger technology developed by Flexiant for use within the MODAClouds project. It has showcased the practical use of this service within a real world example and the importance of such technology within the MODAClouds solution. Detailed is the technology underpinning the triggers technology and example triggers created that are freely available and open sourced.

References

1. Flexiant (2015) Software Features Tour. https://www.flexiant.com/flexiant-cloud-orchestrator/
2. Flexiant (2015) 3rd Party Plugins. https://www.flexiant.com/plugins/about-plugins/
3. Flexiant (2016) Flexiant Cloud Orchestrator Developer Guide. http://docs.flexiant.com/display/DOCS/Flexiant+Cloud+Orchestrator+Developer+Guide
4. Ierusalimschy R, de Figueiredo LH, Celes W (2006) Lua 5.1 Reference Manual. http://www.lua.org/manual/5.1/
5. Flexiant (2016) Introduction to Jade APIs. http://docs.flexiant.com/display/DOCS/Introduction+to+Jade+APIs
6. IETF (2005) A Universally Unique IDentifier (UUID) URN Namespace. https://www.ietf.org/rfc/rfc4122.txt