EPJ Web of Conferences **214**, 08029 (2019) https://doi.org/10.1051/epjconf/201921408029
CHEP 2018

Notifications workflows using the CERN IT central messaging infrastructure

Zhechka Toteva[1,*], *Darko* Lukic[2], and *Lionel* Cons[1]

[1]CERN, 1, Esplanade des Particules, 1211 Geneva 23, Switzerland
[2]EPFL, CE 3 316 (Centre Est), Station 1, CH-1015 , Switzerland

Abstract. In the CERN IT agile infrastructure (AI), Puppet, the CERN IT central messaging infrastructure (MI) and the Roger application are the key constituents handling the configuration of the machines of the computer centre. The machine configuration at any given moment depends on its declared state in Roger and Puppet ensures the actual implementation of the desired configuration by running the Puppet agent on the machine at regular intervals, typically every 90 minutes. Sometimes it is preferable that the configuration change is propagated immediately to the targeted machine, ahead of the next scheduled Puppet agent run on this machine. The particular need of handling notifications in a highly scalable manner for a large scale infrastructure has been satisfied with the implementation of the CERNMegabus architecture, based on the ActiveMQ messaging system. The design and implementation of the CERN-Megabus architecture are introduced, followed by the implementation of the Roger notification workflow. The choice of ActiveMQ is analysed and the message flow between the Roger notification producer and the CASTOR, EOS, BATCH and Load Balancing consumers are presented. The employment of predefined consumer modules in order to speed up the on-boarding of new CERN-Megabus use cases is also described.

1 Introduction

Messaging enables asynchronous communication between services in a highly reliable and configurable manner. It is designed for loosely coupled architectures where producers and consumers do not need to know about each other [1]. Also, it offers instant communication, which is a significant improvement for services that do HTTP polling. However, using messaging may become unnecessary complicated, as it turned out in our use case with Roger[2].

Roger is an in-house developed tool, that manages the application state and the alarm masking for every machine in the CERN IT AI world. Previously, locally installed RabbitMQ [3] message brokers on the Roger servers were used to notify other affected services about Roger state changes. The two biggest customers of the Roger notifications are CASTOR [4] and EOS [5], that have the requirement to change the read/write mode of the tapes (disks respectively) as soon as the Roger state changes of a CASTOR/EOS worker node.

Despite the flexibility to run a configuration tailored to the Roger notifications use case, locally managed message brokers require significant extra support. In order to ensure reliable

*e-mail: Zhechka.Toteva@cern.ch

and scalable notification handling of Roger state changes, the decision was taken to switch to use the CERN IT MI instead [6].

A lot of similarities were found in the described EOS/CASTOR use cases and some other services that require prompt update after another service is changed. These similar needs inspired the birth of the CERNMegabus project.

2 CERNMegabus architecture

CERNMegabus is a service that provides instant messaging communication between services. Its architecture is based on the publisher-consumer model and utilises the CERN IT MI (Figure 1).

Figure 1. CERNMegabus architecture

The publisher and the consumer services comprise building blocks that are configured with Puppet[7] and use the Python libraries, **python-megabus**, specially developed for the CERNMegabus project. The **python-megabus** code has been developed with the idea to be handed to the scientific community, by reducing the coupling with the CERN specific configuration. The CERN specific configuration is handled by Puppet manifests as it is done for all other services in the CERN Computer Centre (CC).

The next paragraphs present in detail the CERNMegabus components, some of which have evolved quite a lot since CERNMegabus is running on all Puppet-managed machines in the CERN CC.

2.1 python-megabus

The **python-megabus** library was developed mainly in order to provide an abstraction on top of the CERN AI and the STOMP protocol. The library consists of two user-facing Python classes, *Publisher* and *Consumer*, that automatically find their configuration file. These classes can be instantiated and configured in any customer Python code. Roger was the first service to profit from CERNMegabus when publishing the Roger state update to the central IT message brokers (Table 1).

The *Publisher* class provides an abstraction of the communication between Service "A" and the CERN IT ActiveMQ message brokers (Step 2 on the Figure 1). The *Consumer* class can be configured and used in customer's Python code in a similar way. This class provides an abstraction of the communication between the CERN IT ActiveMQ message brokers and Service "B" (Steps 1 and 3), leaving the processing of the action (Step 4) to the customer's code. In order to include as well the action handling in the abstraction, initially CERNMegabus used *stompclt* [8] in the consumer's part of the workflow.

After releasing CERNMegabus in production running on more than forty thousand virtual and physical Puppet-managed machines, there was a change introduced to the project - the stompclt part was replaced with a new CERN-developed Python equivalent of it, called *megabusclt*. Apart from having all the code in a single programming language, this new tool mitigated a problem related to the NET6 SSL Perl library. The *megabusclt* tool is distributed in the **python-megabus** library.

teigi/message.py

```
import megabus
…
def send_msg(message, hostname):
  try:
    bus = megabus.Publisher('roger')
    bus.send(message, extension=top_hostgroup,
hostgroup=full_hostgroup, hostname=hostname)
  except Exception as e:
    …
```

activemq-publisher-roger.conf

```
[client]
auth_method = password
server = cernmibroker.cern.ch
port = 12345
destination = roger.notification.hostgroup
destination_type = topic
user = produceruser
pass = thatisasecret
use_multiple_brokers = true
```

Table 1. Roger sending state update via CERNMegabus to the CERN IT MI

2.2 Puppet configuration

All the **python-megabus** user-facing classes and tools are highly configurable with the help of configuration files, one example of which was presented in Table 1. These configuration files are produced by Puppet. Table 2 shows how *activemq-publisher-roger.conf* is created.

manifests/common/server_leveldb.pp

```
::cernmegabus::client::publisher{'roger':
   server           => $activemq_server,
   port             => $activemq_port,
   user             => $activemq_producer,
   pass_key         => 'activemq_user',
   destination      => 'roger.notification.hostgroup',
   destination_type => 'topic',
   owner            => $teigi_user,
}
```

templates/clt/activemq-publisher.conf.erb

```
[client]
auth_method = <%= @auth_method %>
server = <%= @server %>
port = <%= @port %>
destination = <%= @destination %>
destination_type = <%= @destination_type %>
user = <%= @user %>
pass = %TEIGI__<%= @pass_key %>__%
use_multiple_brokers = <%= @use_multip.. %>
```

Table 2. Puppet configuration of roger CERNMegabus Publisher

The *server_leveldb.pp* Puppet manifest uses the CERN Puppet resource *::cernmegabus::client::publisher* to provide the values for the ERB Puppet[9] template *activemq-publisher.conf.erb*. The later manages the content of *activemq-publisher-roger.conf*.

The *Consumer* can be configured and embedded in customer Python code in a similar way, using a corresponding Puppet manifest and a Puppet template. The parameters are mostly the same, except that there are two additional CERN specific consumer parameters, namely *hostgroup_selector* and *host_selector*, which will be covered in detail in a later section of this paper.

Megabusclt requires the same parameters as the *Consumer* class and the extra ones for configuring the actions to be taken.

3 CERNMegabus features

CERNMegabus service provides variety of configurable features that make significant impact to any message-driven communication. The next couple of sections of this document are dedicated to the most appealing features for the Python-base service-oriented systems.

3.1 Authentication and authorisation

CERNMegabus supports two authentication schemes:

- x509 certificate-based[10]
- basic authentication that requires an user name and a password, which are locallly managed in the ActiveMQ message brokers

The authentication schema is set per topic/queue by the CERN IT MI service manager. In order that CERNMegabus connects successfully a publisher/consumer to a ActiveMQ topic/queue, the correct authentication schema should be provided with correct credentials.

CERNMegabus defaults to x509 when choosing the authentication schema. If basic authentication is explicitly selected, the user name and the password have to be supplied. The password is expected to be stored in TBAG[11] and retrieved from there and stored on the client node by Puppet.

3.2 Publisher-consumer model

Different message brokers offer different approaches for providing high availability architecture[3]. CERN IT MI provides multiple independent brokers[12] available behind a DNS Load Balancing (LB) alias[13].

With the LB approach without explicit replication of each message on every broker, the publisher-consumer models can be individually designed for the needs of each application. The choice depends on the ratio between the number of publishers (n(pub)) and number of consumers (n(cons)). The goal is to minimise the number of network connections needed to guarantee that a message sent by any publisher can reach all the affected consumers. This algorithm results in two possible publisher-consumer models:

- if n(pub) >= n(cons): Publish to one (random) message broker behind the LB alias and subscribe to consume from all message brokers.
- if n(pub) < n(cons): Publish to all message brokers behind the LB alias and subscribe to consume from one (random) message broker.

CERNMegabus brings value here by proposing transparent dereferencing of the DNS LB alias for the consumer and for the producer by a single flag *use_multiple_brokers* in the **python-megabus** library configuration (see activemq-publisher-roger.conf in Table 1). This flag is also configurable via Puppet (see Table 2). Depending if the *use_multiple_brokers* flag is set for the publisher or for the consumer, the message is sent to all (or respectively is read from all) message brokers behind the DNS LB alias.

3.3 Puppet plugins

Plugins are the essential part of the CERNMegabus service. They demonstrate how powerful, but still simple-to-use, notification handling in a big computer centre can be. The CERNMegabus plugins are available as Puppet resources and in most cases they need only a couple of parameters to be initialised, e.g. a command that should be executed on a message arrival.

These plugins were initially built on top of stompclt Puppet resources (Figure 2), and later one migrated to use megabusclt Puppet resources.

3.3.1 teigiclt_roger_actions plugin

There are different consumers of Roger state changes, but the one installed on all 40 thousand machines in the CERN CC listens to updates of the Roger state of the machine itself. This is achieved with only one line of code:

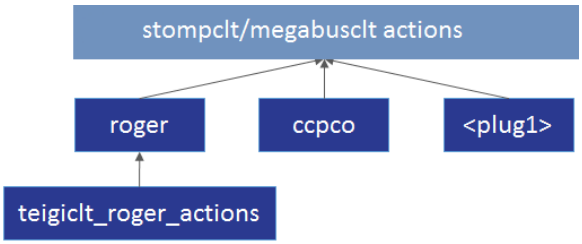

Figure 2. Inheritance of CERNMegabus plugins

```
include ::cernmegabus::plugins::teigiclt_roger_actions
```

The *teigiclt_roger_actions* plugin inherits from the *roger* one, initialising only the *on_change_command* parameter with a path to a bash script residing on the file system of the machine.

```
::cernmegabus::plugins::roger { 'file':
  on_change_command     => '/usr/bin/roger_actions',
}
```

3.3.2 Roger plugin

The *roger* plugin inherits from the *cernmegabus::action* plugin by passing the already initialised *on_change_command* parameter. The *on_change_command* parameter goes to the template *cernmegabus/plugins/roger.sh.erb* that forms the content of the bash script (*/usr/libexec/megabusclt-actions/roger-file*) that will end up in the client file system. This bash script will be listed in the megabusclt configuration as the action to be executed:

```
define cernmegabus::plugins::roger (
  String $on_change_command, ...
) {
  cernmegabus::action { "roger-${title}":
    host                 => $::cernmegabus::roger_server,
    destination          => $::cernmegabus::roger_dest,
    filters              => {'hostname' => "mymachine.cern.ch"}
    use_broker_filtering => true,
    command              =>
          template('cernmegabus/plugins/roger.sh.erb'),
  }
}
```

3.3.3 CCPCO plugin

The *ccpco* plugin handles notifications related to CERN Computer Centre Power Cut Orchestration (CCPCO) workflow. The default action to be taken in case of a power cut on all machines in the computer centre is declared with only one line of code:

```
cernmegabus::plugins::ccpco { 'base': }
```

This plugin hides the detail that the action to be taken is sending an email to the responsible people of the machine. Once the IT management approves the CCPCO workflow for production, the same plugin gives the possibility the default action to be switched transparently to shutdown. The *ccpco* plugin provides several predefined actions like logging to a log file, shutting down or sending an email. The service manager is also offered the option to execute any user-defined bash script that is stored on the machine file system.

Due to the diverse requirements that different services have in case of a power cut, the default action can be be overwritten by the service managers by a list of actions. Each action is tagged with a starting time, which is represented with the number of seconds after the power cut event. This is implemented by Puppet resource collectors:

```
Cernmegabus::Plugins::Ccpco <| title == 'base' |> {
  power_cut_actions => {
    0   => 'std_log --path=/var/log/powercut.log',
    120 => '/bin/backdata.sh',
    240 => 'std_email --to teigi-admins@cern.ch',
    360 => 'std_shutdown',
  }
}
```

3.3.4 New plugin development

CERNMegabus provides a simple interface for developing new plugins. In order to develop a new plugin, users can use the stompclt::action or the *megabusclt::action* Puppet resource.

3.4 Puppet stompclt resources

Although the stompclt resource has been replaced with the megabusclt one for the purposes of the CERNMegabus service, the users of the stompclt service can still profit from this resource for building the stompclt configuration file. In order to configure the stomcplt daemon, a few Puppet resources are created. The architecture of the *stompclt* resource enables configuration of multiple authentications, subscriptions and actions per stompclt instance. The resource implements reuse of subscriptions and TCP connections whenever it is possible.

3.5 Reuse of TCP connections, subscriptions and selectors

Consumers subscribe to queues or to topics and these subscriptions become more fine-grained by using ActiveMQ message broker-side selectors. These selectors express criteria in SQL-92 syntax that are applied on the header parameters of the messages. Although broker-side selectors reduce the amount of messages being sent to the clients, they can increase the number of subscriptions to the broker significantly in the case of generic topics. Both stompclt and megabusclt provide the choice of broker-side or client-side filtering, by using a simple flag *use_broker_filtering*, which can be configured by Puppet as well. For the moment CERN-Megabus supports only simple client-side filtering, based on *hostgroup* and *hostname*, terms that are specific for the CERN AI world.

4 Use cases

The design of CERNMegabus service was mainly driven by use cases. Despite of their diversity most of the use case fall into one of these three classes of message consuming:

4.1 Consume message affecting my workers

From this class of use cases, CASTOR Roger state listener was the first one migrated to the CERNMegabus service. The following example demonstrates some implementation details. CASTOR LHCb headnodes subscribe to the ActiveMQ message broker on topic "*/topic/roger.hostgroup.castor*" with broker-side filtering hostgroup_selector "*castor-lhcb-diskserver-%*". The broker-side filtering ensures that the CASTOR LHCb nodes are not getting messages for CASTOR CMS disk servers for example. This use case profits from the *roger* Puppet plugin. On message arrival, if there is a change of the Roger application state of the diskserver in question, a command is run to adapt the read/write state of the tapes accordingly.

The other use cases from this class are configured to use CERNMegabus in a similar way, with the similarity that they all rely on the *hostgroup_selector*.

- EOS - uses topic "*/topic/roger.hostgroup.eos*" with broker-side filtering hostgroup_selector "″eos/<instancename>/storage". Due to the complexity of the action that handles the received message, EOS profits from the **python-megabus** library embedded in their customer code and configured with the *::cernmegabus::client::consumer* Puppet resource. The action taken is similar to the CASTOR one, namely to ensure that the read/write mode of the affected node is consistent with the updated Roger state.
- Puppet HAProxy - uses topic "*/topic/roger.hostgroup.punch*" with broker-side filtering hostgroup_selector "″punch/Puppet/ps/v4/%/<h3>". This use case profits from the *roger* plugin on change of the Roger application state from/to "production" to run directlty the HAProxy ctl sub-commands to disable/enable the machine in question.

4.2 Consume messages affecting myself

From this class of use cases, the DNS LB client was the first one migrated to the CERNMegabus service. The following example demonstrates some implementation details.

The DNS LB client is available on all Puppet-managed machines that are members of an LB alias. There are many configurable criteria considered in the decision if a machine is healthy in order to participate in an LB alias. If Roger application state is one of the criteria for healthiness of a machine, it has to be verified by the LB client on regular intervals. Previously, the LB client was querying (polling) the Roger state of a machine directly from the Roger server, and if unavailable if was falling back to the locally cached Roger state stored in *current.yaml* file on the machine. That file was updated on a Puppet agent run, which was between every 1 and 6 hours, depending on the services the machine provides.

CERNMegabus facilitates the services, which run locally on the machine and need up-to-date Roger state, by immediately propagating the Roger state to the current.yaml file, overtaking the next Puppet agent run. This enhancement eliminated the need of the Roger server to be contacted. Another service that will soon profit from the change is the Alert Handler, that collects alarms information for the monitoring infrastructure on every node in the CERN CC.

All these use cases rely on the *host_selector* criteria to be the FQDN of the machine.

4.3 Consume messages affecting everybody

The big use case of this class is the CERN CCPCO workflow, that has been already introduced in this paper with the *ccpco* Puppet plugin. In the CCPCO workflow, we have two machines monitoring the UPS systems in the CERN CC. In case of a power cut, they send (broadcast)

a message to a general topic *topic/ccpco.notification* on which all machines in the CC are subscribed. The presence of a power cut is verified every five seconds and a new message is sent every minute in order to ensure that all machines are notified with the exact time elapsed since the power cut event. It is estimated that the UPS can last for about 20 minutes. When the power is back in time, a new message is broadcast announcing the power back event.

5 Conclusion

A new approach to handle notifications in a highly scalable manner for a large scale infrastructure using Puppet, was presented in this paper. A detailed technical description was given of the CERNMegabus service design based on Python, Puppet, ActiveMQ, and stomcplt/megabusclt. As a result of this developemnt, users are given access to a simple CERNMegabus API that easily handles notifications. CERNMegabus provides means to decrease the load on message brokers by reusing connections, subscriptions and selectors. The success of CERNMegabus service allows for planning to use it for even more intensive daily activities in the CERN computer centre.

References

[1] The WLCG Messaging Service and its Future, L.Cons and M. Paladin, Journal of Physics: Conference Series, **396(3)**, p.032084, (2012).

[2] Alarm masking and application state management tool [software], Available at https://twiki.cern.ch/twiki/bin/view/Main/rogerClient.

[3] RabbitMQ - Distributed RabbitMQ brokers. [software] Available at: https://www.rabbitmq.com/distributed.html [Accessed 21 Jun. 2018].

[4] CASTOR: A Distributed Storage Resource Facility for High Performance Data Processing at CERN, G. Presti, O. Barring, A. Earl, R. Rioja, S. Ponce, G. Taurelli, D. Waldron and M. Santos, 24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007), **9880374**, p.275-280, (2007).

[5] Latest evolution of EOS filesystem, G. Adde, B. Chan, D. Duellmann, X. Espinal, A. Fiorot, J. Iven, L. Janyst, M. Lamanna, L. Mascetti, J. Rocha, A. Peters and E. Sindrilaru, Journal of Physics: Conference Series, **608**, p.012009, (2015).

[6] Enterprise Messaging Solutions Technical Evaluation, L. Cons and M. Paladin, European Middleware Initiative, Available at http://cern.ch/go/w9Qd, (2011).

[7] Scaling Agile Infrastructure to People, B. Jones, G. McCance, S. Traylen and N. Arias, Journal of Physics: Conference Series, **664(2)**, p.022026, (2015).

[8] stompclt, L. Cons, [software] GitHub. Available at: https://github.com/cern-mig/stompclt [Accessed 21 Jun. 2018].

[9] Puppet, [software], Available at: https://puppet.com/.

[10] Overview of Certification Systems: X.509, PKIX, CA, PGP SKIP, E. Gerck, The Bell Newsletter, **ISSN 1530-048X, Vol. 1**, p.8, (2007).

[11] Managing secrets with TBAG [software], Available at https://configdocs.web.cern.ch/configdocs/secrets

[12] Modern Messaging for Distributed Systems, L. Magnoni, Journal of Physics: Conference Series, **608**, p.012038, (2015).

[13] DNS load balancing in the CERN cloud, I. Reguero and L. Lobato, Journal of Physics: Conference Series, **898**, p.062007, (2017).