Publish/Subscribe Messaging at XSEDE

Version Jan. 27, 2015

Publish/Subscribe Messaging Overview

XSEDE has deployed publish/subscribe a messaging service to distribute information gathered from XSEDE resources to consumers that wish to receive this information. A simplified version of publish/subscribe messaging model deployed by XSEDE is:

- Consumers subscribe for the information they wish to receive from the messaging service and then wait for the messaging service to send information to them.
- Producers of information publish information to the messaging service as they generate it.
- The Messaging service receives messages containing information from publishers, determines which consumers want to receive this information, and forwards these messages to those consumers.

An important characteristic of this approach is that information is delivered very quickly from where it is produced to the tools that use the information.

At the current time, static and dynamic descriptions of XSEDE resources are being published as GLUE v2.0 documents (described below) and resource and service status is being published as Inca test results. This information is primarily of interest to the XSEDE portal and XSEDE science gateways, but it is possible for others to also subscribe for it. While some of the published information is not sensitive and is available to anonymous consumers, a consumer of other types of information must authenticate to the messaging service, typically via an X.509 certificate. All publishers of information must authenticate, again typically via an X.509 certificate.

XSEDE and Publish/Subscribe Messaging

Figure 1 below provides details of the XSEDE publish/subscribe messaging deployment. The main part of the deployment is two RabbitMQ [RabbitMQ] messaging services. Two services are deployed for fault tolerance and they are configured in a clustered (or active/active) configuration. Producers and consumers can connect to either service and we recommend that producers and consumers be configured to try one service and if that connection fails, to try the other service.

These services support the standardized Advanced Message Queuing Protocol (AMQP) [AMQP-0.9.1] and there are a variety of AMQP client libraries available [RabbitMQ-Clients]. These clients are used both to publish messages and to subscribe for messages.

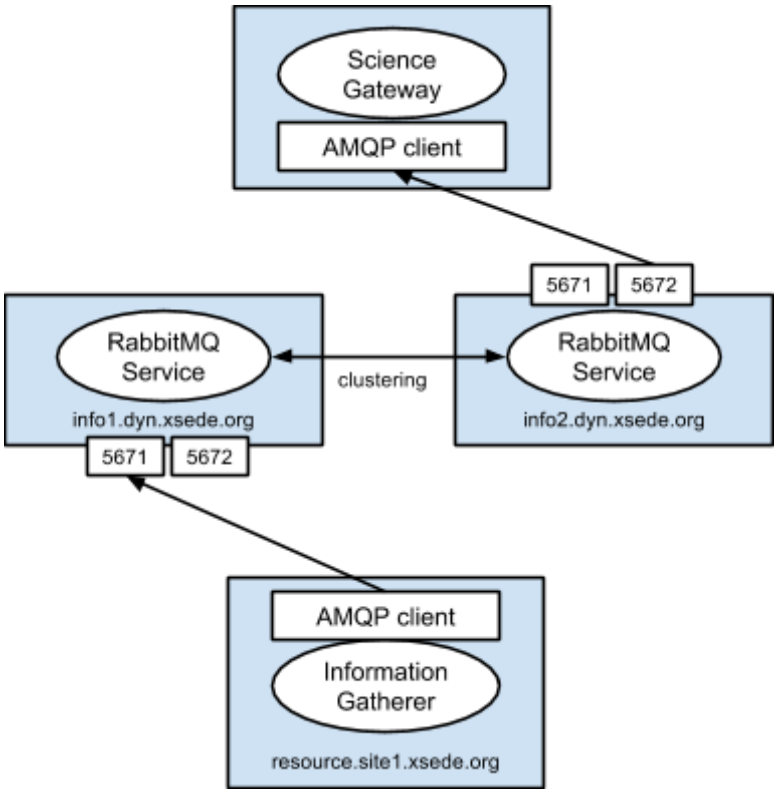

Figure 1. XSEDE publish/subscribe messaging deployment.

Using Publish/Subscribe Messaging

The RabbitMQ services can be contacted over either an insecure TCP connection (port 5672) or a secure SSL/TLS connection (port 5671). Insecure connections are only used to subscribe anonymously (username: guest, password: guest) for public XSEDE information. Secure connections are used for all publishing and to subscribe for non-public XSEDE information. The RabbitMQ services have SHA-2 certificates from the Incommon RSA Server CA. To authenticate the RabbitMQ services, your client will therefore need the certificates in this CA certificate chain. The CA certificates you need are listed on the InCommon Cert Types page.

An XSEDE-provided RabbitMQ username/password can be used to authenticate over a secure connection, but the preferred way is to have the client present an X.509 certificate. This certificate must come from one of the certificate authorities accepted by XSEDE. To enable either authentication method, please submit an XSEDE ticket.

The RabbitMQ website provides a number of tutorials [RabbitMQ-Tutorials], but listing 1 below contains a very simple example of how to receive messages containing Inca test results and print them to the screen. This example uses the RabbitMQ Java client library and you will need to download it and include it in your classpath.

The example subscribes anonymously to the ‘inca’ exchange in the ‘xsede’ virtual host of the RabbitMQ service running on ‘info1.dyn.xsede.org’. When creating the connection factory, no authentication information is provided so the default user ‘guest’ and password ‘guest’ is used. An exchange acts as a mailbox where messages are published to - the Inca testing service is publishing to the ‘inca’ exchange. Each message consists of a set of data and a routing key - a tag providing some metadata. In the case of Inca, the routing key contains success or failure, the name of the test performed, and the name of the component that was tested. A virtual host is a logical container for exchanges and queues.

Messages are routed from exchanges to queues where they are stored until a consumer is ready to receive them. In the example, a queue is created and is bound to the ‘inca’ exchange using a filter of ‘#’, which indicates that the consumer wants to receive messages with any routing key. This filter can be used to specify that only specific messages should be sent to the consumer (e.g. from a specific resource, a specific test).

```
import java.io.IOException;
import java.util.List;

import com.rabbitmq.client.*;

public class SubscribeInca {
    public static void main(String[] argv) throws Exception {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("info1.dyn.xsede.org");
        factory.setPort(5672);
        factory.setVirtualHost("xsede");
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        String queueName = channel.queueDeclare().getQueue();
        channel.queueBind(queueName,"inca","#");
        channel.basicConsume(queueName,true,new IncaConsumer());

        System.out.println("**** To exit press CTRL+C ****");
        while(true) {
            try {
                Thread.sleep(1*1000);
            } catch (InterruptedException e) { }
        }
    }
}

class IncaConsumer implements Consumer {
    public IncaConsumer() { }

    public void handleConsumeOk(String consumerTag) { }
```

```
    public void handleCancel(String consumerTag) { }

    public void handleCancelOk(String consumerTag) { }

    public void handleShutdownSignal(String consumerTag,
                                     ShutdownSignalException sig) { }

    public void handleRecoverOk(String consumerTag) { }

    public void handleDelivery(String consumerTag,
                               Envelope envelope,
                               AMQP.BasicProperties properties,
                               byte[] body) throws IOException {
        System.out.println(envelope.getRoutingKey());
        System.out.println(new String(body));
    }
}
```

Listing 1. Simple example of subscribing for Inca testing messages.

References

[AMQP-0.9.1] Advanced Message Queuing Protocol 0.9.1. http://www.amqp.org/specification/0-9-1/amqp-org-download

[RabbitMQ] RabbitMQ: Messaging that just works. http://www.rabbitmq.com

[RabbitMQ-Clients] Clients and Developer Tools. http://www.rabbitmq.com/devtools.html

[RabbitMQ-Tutorials] RabbitMQ Tutorials. http://www.rabbitmq.com/getstarted.html

GLUE v2.0 at XSEDE

GLUE v2.0 Overview

The XSEDE user portal, science gateways, and end users all need static and dynamic information about XSEDE. This information is used to inform users about the configuration and status of XSEDE, help users and gateways select where to run, and let user and gateways track their use of XSEDE.

GLUE v2.0 [GLUE2] is a standard way to represent information about resources, services, and software developed in the Open Grid Forum. This information is used by the XSEDE user portal, science gateways, and end users to understand that static and dynamic state of XSEDE. In addition to a data model, renderings of that data model have been defined for various representations, such as the JSON rendering [GLUE2-JSON] used by XSEDE.

XSEDE is deploying software called the Information Publishing Framework [IPF] to gather information about XSEDE resources, to represent that information as GLUE v2.0 JSON, and to publish that information via publish/subscribe messaging.

XSEDE and GLUE v2.0

XSEDE is currently using GLUE v2.0 to represent compute resources, jobs, and installed software. This information is being published by all compute resources to the XSEDE publish/subscribe messaging service described above. GLUE v2.0 information is published to four exchanges in the 'xsede' virtual host. These exchanges are shown in Table 1 along with the information type and whether authentication is required.

Table 1. RabbitMQ exchanges.

Exchange	Information Type	Authentication Required
glue2.compute	Compute Resource Description	No
glue2.computing_activities	Queue State	Yes
glue2.computing_activity	Job Information	Yes
glue2.applications	Module Definitions	No

Every few minutes, XSEDE compute resources publish a summary of their configuration and status to the glue2.compute exchange. The routing keys for each message is the the XSEDE name of the resource - typically SYSTEM.SITE.xsede.org. The content of each message will be similar to https://github.com/OGF-GLUE/JSON/blob/master/examples/compute.json and contains:

- A summary of the batch scheduler (ComputingManager, ComputingService)
- A summary of the queues (ComputingShare)
- A summary of the hardware (ExecutionEnvironment)

Every few minutes, XSEDE compute resources publish a description of the jobs that they are managing to the glue2.computing_activities exchange. The routing keys for each message is

the the XSEDE name of the resource and the content of each message will be similar to https://github.com/OGF-GLUE/JSON/blob/master/examples/activities.json and contains a list of job descriptions (ComputingActivity).

As jobs change state, XSEDE compute resources publish information about each job individually to the glue2.computing_activity exchange. The routing keys for each message are JOB_ID.LOCAL_USER_NAME.SYSTEM.SITE.xsede.org. Each message contains the description of a single job (ComputingActivity) that has just been submitted or changed state.

Finally, approximately every hour, XSEDE compute resources publish information about the modules installed on the system to the glue2.applications exchange. The routing keys for these messages are the XSEDE name of the resource. Each message will be similar to https://github.com/OGF-GLUE/JSON/blob/master/examples/modules.json and will contain descriptions of software packages (ApplicationEnvironment) and information on the module that loads that software into a user environment (ApplicationHandle).

Running GLUE v2.0

To access the GLUE v2.0 published by XSEDE, Listing 1 can be used as a pattern. For the glue2 exchanges that do not require authentication (listed in Table 1), simply replace the “inca” in the queueBind call in Listing 1 with the name of the exchange (“glue2.compute” or “glue2.applications”).

For the glue2 exchanges that do require authentication, see the examples on https://www.rabbitmq.com/ssl.html for how to configure your ConnectionFactory. As described in the Publish/Subscribe Messaging documentation, your client will need to trust the CA certificates listed on the InCommon Cert Types page and your username/password or X.509 certificate DN will need to be authorized in the XSEDE RabbitMQ services.

The general workflow for handling messages from glue2 is to receive them, parse the JSON documents into structures or objects, and then use the information in the structures or objects in tool-specific ways. A simple way to do this is to use a JSON library to generate generic maps and lists with values in them. This approach is convenient in dynamically typed languages such as Python.

Another approach that may be more convenient for statically typed languages is to parse the JSON documents into custom GLUE v2.0 objects. An example of this approach is available from https://github.com/OGF-GLUE/JSON/tree/master/examples/java. This example defines a set of Plain Old Java Objects and uses the Jackson library to parse a JSON GLUE v2.0 document into these objects. A simple example progam that shows how to use these objects is located at https://github.com/OGF-GLUE/JSON/blob/master/examples/java/src/Glue2Pojo.java.

References

[GLUE2] GLUE Specification v2.0. http://www.ogf.org/documents/GFD.147.pdf

[GLUE2-JSON] JSON Rendering of the GLUE 2.0 Specification. https://github.com/OGF-GLUE/JSON

[GLUE2-JSON-example] https://github.com/OGF-GLUE/JSON/tree/master/examples/java

[IPF] Information Publishing Framework. https://bitbucket.org/wwsmith/ipf