Apps-Script-PubsubApp-Library

An Apps Script library for the Google Cloud Pub/Sub Service

Cloud Pub/Sub is a tool for many-to-many asynchronous messaging between independent applications and users. It decouples publishers, topics, subscriptions, and subscribers to achieve maximum flexibility. At the most basic Publishers can post messages to a topic cue. Subscribers can watch their own instance of the topic cue and read and acknowledge receipt of the messages. The service allows for both push and pulling of the subscription cue. For in depth documentation refer to the official pub/sub docs at: https://cloud.google.com/pubsub/docs

Setup your project to use the PubsubApp library:

1) Add the library Mk1rOXBN8cJD6nl0qc9x5ukMLm9v2IJHf or use the source code from the src folder
2) Add an Oauth2 service with either scope:
   https://www.googleapis.com/auth/pubsub
   https://www.googleapis.com/auth/cloud-platform

   If you are using a service account or managing domain accounts you can use my GSApp library:
   https://github.com/Spencer-Easton/Apps-Script-GSApp-Library

   If you are adding this to a client faced application running on their own account credentials use:
   https://github.com/googlesamples/apps-script-oauth2

3) Enable the pub/sub api in the developers console.

Initialize the PubsubApp Service

The first thing is to set the token service for the PubsubApp. You can do this with the setTokenService(function) method. The function passed to setTokenService will be invoked every time the library requests a token. It is the responsibility of the passed function to check for token freshness and request new tokens as needed as PubsubApp will not do this.

Example with the GSApp

```
function myFunction(){
  PubSubApp.setTokenService(getTokenService());
}

function getTokenService(){
  var jsonKey =
JSON.parse(PropertiesService.getScriptProperties().getProperty("jsonKey"));
  var privateKey = jsonKey.private_key;
  var serviceAccountEmail = jsonKey.client_email;
  var sa = GSApp.init(privateKey,
['https://www.googleapis.com/auth/pubsub'], serviceAccountEmail);
  sa.addUser(serviceAccountEmail)
      .requestToken();
  return sa.tokenService(serviceAccountEmail);
}
```

Example with googlesamples/Oauth2

```
function myFunction(){
   PubSubApp.setTokenService(function(){
                    return getPubsubService().getAccessToken();
    });
  }

function getPubsubService() {
    return OAuth2.createService('pubSub')
         .setAuthorizationBaseUrl('https://accounts.google.com/o/oauth2/auth')
         .setTokenUrl('https://accounts.google.com/o/oauth2/token')
         .setScope('https://www.googleapis.com/auth/pubsub')
          ...
          ...
  }
```

The PubsubApp Library has three services:

1) PublishingApp
2) SubscriptionApp
3) PolicyBuilder

PublishingApp

The PublishingApp service all you to manipulate topics and post messages to those topics.

Constructor

PublishingApp(string apiProjectId) -> new PublishingApp serivce

Constructor for the service. Returns a new instance of the service. apiProjectId is the Developer Console Project ID where the pubsub api is enabled

Methods

getTopics()	**-> array [{topic resource},...]**
getTopicName(string frendlyTopicName)	**-> string fullTopicName**
getTopic(string frendlyTopicName)	**-> object new Topic**
newTopic(string frendlyTopicName)	**-> object new Topic**
deleteTopic(string frendlyTopicName)	**-> object this PublishingApp**
newMessage()	**-> object pubsubMessage**

Topic Object Methods

getIamPolicy() -> object IamPolicy
setIamPolicy(policyObject) -> object IamPolicy
getSubscriptions() -> array [subscriptionName, ...]
publish(pubsubMessage) -> array [messageIds, ...]
getName() -> string fullTopicName

SubscriptionApp

The SubscriptionApp service allows you to manipulate subscriptions to topics and to read and acknowledge receipt of messages.

Constructor

SubscriptionApp(string apiProjecyId) -> new SubscriptApp Service

Methods

getSubscriptions() ->array [{subscription resource}, ...]
getSubscription(subscriptionName) -> object new Subscription
deleteSubscription(subscriptionName) -> object this SubscriptionApp
newSubscription(subscriptionName,topicName,webhookUrl)
-> object new Subscription

Subscription Object Methods

setIamPolicy(object IAmPolicy) -> object IAmPolicy
getIamPolicy() -> object IAmPolicy
pull(number maxCount, boolean autoAcknowlage)
-> object (autoAck) ? array [pubsubMessage, ...] : array [ReceivedMessage, ...]
ack(array ackIds) -> object this Subscription
modifyAckDeadline() -> object this Subscription

PolicyBuilder

The policyBuilder service allows you to manipulate the access policies to both topics and subscriptions. This is handled through the standardized Google Cloud IAmPolicy object. You can read more at: https://cloud.google.com/iam/

Constructor

policyBuilder() -> object new PolicyBuilder Service

Methods

newPolicy() -> object new Policy
editPolicy(object IAmPolicy) -> object new Policy

Policy Object Methods

addOwner(string memberType, string memberName) -> object this Policy
addEditor(string memberType, string memberName) -> object this Policy

addViewer(string memberType, string memberName)	-> object this Policy
addPublisher(string memberType, string memberName)	-> object this Policy
addSubscriber(string memberType, string memberName)	-> object this Policy
removeOwner(string memberName)	-> object this Policy
removeEditor(string memberName)	-> object this Policy
removeViewer(string memberName)	-> object this Policy
removePublisher(string memberName)	-> object this Policy
removeSubscriber(string memberName)	-> object this Policy
getPolicy()	-> object IAmPolicy

Member Types

"ALLUSERS"
"USER"
"SERVICEACCOUNT"
"GROUP"
"DOMAIN"

Examples

Setting up a new topic

```
function CreateTopic(topicName) {
   var topic;
   PubSubApp.setTokenService(getTokenService());
   var pubservice = PubSubApp.PublishingApp(PROJECTID);
   try{topic = pubservice.newTopic(topicName)}
   catch(e){topic = pubservice.getTopic(topicName);}
  return topic;
}
```

Modify a topics IAM Policy

```
 function alterTopicPolicy(topicName){
   PubSubApp.setTokenService(getTokenService());
   var mytopic = PubSubApp.PublishingApp(PROJECTID).getTopic(topicName);
   var policy = mytopic.getIamPolicy();
   var newPolicy = PubSubApp.policyBuilder().editPolicy(policy);
   newPolicy.addSubscriber("SERVICEACCOUNT",
                                            "gmail-api-push@system.gserviceaccount.com");
  return mytopic.setIamPolicy(newPolicy.getPolicy());
 }
```

More IAM Policy examples

```
function testPolicyBuilder(){
 var pBuilder = PubSubApp.policyBuilder();
 var newPolicy = pBuilder.newPolicy();
 newPolicy.addOwner("USER","test@example.com");
 var policy2 = PubSubApp.policyBuilder().editPolicy(newPolicy.getPolicy())
 policy2.addSubscriber("DOMAIN","example.com")
           .removeOwner(“USER”,”test@example.com”)
           .addViewer(“GROUP”,”myGroup@example.com”)
          .addPublisher(“SERVICEACCOUNT”,”serviceaccount12345@googleusercontent.com”)
 Logger.log(policy2.getPolicy())
 }
```

Create a new Subscription

```
// If webhookUrl is null PULL delivery will be used
// topicName must be the full name ie project/myProjectId/topics/myTopic
// You can get this with the PublingApp.getTopicName() method
function CreateSubscription(subscriptionName,topicName,webhookUrl){
  var sub;
  PubSubApp.setTokenService(getTokenService());
  var subService = PubSubApp.SubscriptionApp(PROJECTID);
  try{sub = subService.newSubscription(subscriptionName,topicName,webhookUrl)}
  catch(e){sub = subService.getSubscription(subscriptionName,topicName,webhookUrl)}
  return sub;
}
```

Post A Message

```
function sendMessage(){
   PubSubApp.setTokenService(getTokenService());
   var pub = PubSubApp.PublishingApp(PROJECTID);
   var mytopic = pub.getTopic('mytopic');
   var message = pub.newMessage();
   message.data = Utilities.base64Encode('This is the test Message!');
   message.attributes["Time"] = new Date();
   message.attributes["publisher"]  = Session.getActiveUser().getEmail();
   Logger.log(mytopic.publish(message));
}
```

Pull A Message with Auto Acknowledge

```
function getMessage(){
  var NumOfMessagesToPull = 1;
  PubSubApp.setTokenService(getTokenService())
  var sub = PubSubApp.SubscriptionApp(PROJECTID);
  var mysub = sub.getSubscription("mySub");
  var message =  mysub.pull(NumOfMessagesToPull);
  if(message.length > 0){
    for(var i=0; i < message.length;i++){
      Logger.log(Utilities.newBlob(Utilities.base64Decode(message[i].data)).getDataAsString());
      Logger.log(message[i].attributes)
    }
  }
}
```

Pull Message with manual Acknowledge

```
function getMessage(){
  var ackIds = [];
  var NumOfMessagesToPull = 3;
  PubSubApp.setTokenService(getTokenService())
  var sub = PubSubApp.SubscriptionApp(PROJECTID);
  var mysub = sub.getSubscription("mySub");
  var message =  mysub.pull(NumOfMessagesToPull,false);
  if(message.length > 0){
    for(var i=0; i < message.length;i++){
        Logger.log(Utilities.newBlob(Utilities.base64Decode(message[i].message.data))
                                  .getDataAsString());
        Logger.log(message[i].message.attributes);
          ackIds.push(messages[i].ackId);
     }
     mysub.ack(ackIds);
  }
}
```

PUSHed Message Processing

```
function doPost(e) {
  var postBody = JSON.parse(e.postData.getDataAsString());
  var messageData =
Utilities.newBlob(Utilities.base64Decode(postBody.message.data)).getDataAsString();
  var ss = SpreadsheetApp.openById('...')
                 .getSheetByName("Log");
  ss.appendRow([new Date(), messageData, JSON.stringify(postBody,undefined,2)])
  return 200;
}
```