Rapidly Building Apps on the Cloud

udemy.com/building-highly-scalable-apps-on-the-cloud/learn/v4/t/lecture/2963916

Services allow you to quickly add capabilities to your application.In this exercise you will create 2 applications that communicate using a messaging service.

We will use Node.js as the programming language for this exercise and you can choose to use Eclipse, the command line or DevOps Services to complete this exercise.

The basic application for both the sender and receiver is shown below. The application should be in a file called **app.js**:

```
var express = require('express');  var appport = process.env.VCAP_APP_PORT || 3000;
//Setup web server  var app = express();  app.listen(appport);
```

This application uses the express framework, so you need to add the dependency to the **package.json** (Substitute text within '<>' with appropriate values):

```
{      "name": "<app name>",      "version": "0.0.1",      "description": "<app
description>",      "dependencies": {          "express" : "4.9.3"      },
"scripts": {          "start": "node app.js"      }  }
```

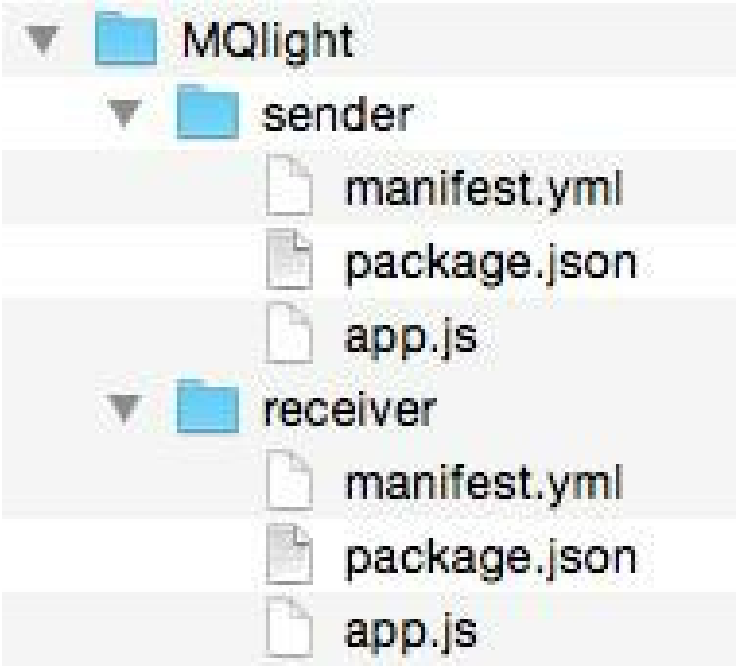
A suggested **manifest.yml** file for the receiver applications is:

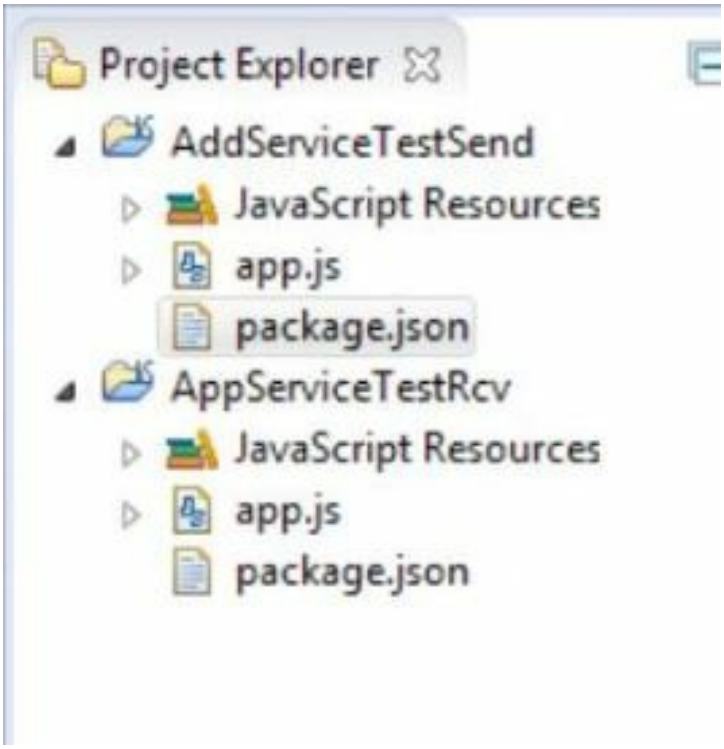
```
---  applications:   - name: MQLrec  host: MQLrec-${random-word}  path: .
instances: 1  memory: 128M
```

For the sender application use the name MQLsnd and host MQLsnd-${random-word}

Create 2 folders or projects (as shown below), one called 'sender' and the other called 'receiver' and create the app.js, manifest.yml and package.json in each folder / project using the sample content shown above.

In case using Node.js projects, the app.js and package.json files will be created with sample content and can be modified with content given above and further below. The manifest file can be created while deploying the application. We can also rename the application while deploying and as such the folder names will not show as name of the application.





Deploy an instance of the MQLight service using either the Bluemix UI or the command line utility.

At the Bluemix Web UI you can deploy an instance of the MQLight from the ‘Web and Application’ section

When deploying the service use the name MQLight and leave the service unbound.

At the command line use the following command:

```
cf cs mqlight standard MQLight
```

To access the service from the applications we need to import a library to the node application (this information can be accessed from the service documentation).

Modify the package.json to require the additional package.Add the additional package to the dependencies section in both the sender and receiver applications:

```
"dependencies": {      "mqlight":"1.0.2014091001",      "express"
: "4.9.3"  },
```

We can now add the package into the applications (again this needs to be done in both the sender and receiver app.js files).At the top of the application pull in the mqlight package and assign it to a variable to be able to access the features in the package:

```
var mqlight = require('mqlight');
```

To be able to connect to the MQLight service the environment variable VCAP_SERVICE

needs to be parsed to obtain the connection information.The code below shows how to allow for both local deployment and Bluemix deployment :

```
if (process.env.VCAP_SERVICES) {      //running in bluemix      var credentials =
JSON.parse(process.env.VCAP_SERVICES)['mqlight'][0].credentials;      opts = {
user:credentials.username,           password: credentials.password,
service:credentials.connectionLookupURI      };  } else {      //running locally
opts = {service:'amqp://localhost:5672'};  }
```

The code above needs to be added to both the sender and receiver application app.js.The start of the applications should now contain:

```
var mqlight = require('mqlight');  var express = require('express');  var appport =
process.env.VCAP_APP_PORT || 3000;  //get connection settings for MQLight  if
(process.env.VCAP_SERVICES) {      //running in bluemix      var credentials =
JSON.parse(process.env.VCAP_SERVICES)['mqlight'][0].credentials;      opts = {
user:credentials.username,           password: credentials.password,
service:credentials.connectionLookupURI      };  } else {      //running locally
opts = {service:'amqp://localhost:5672'};  }
```

To complete the exercise the sender and receiver applications will diverge from here.Starting with the sender application.

The sender application will publish a message each time a web request is received.To connect to the MQLight service and publish a message on starting the connection add the following code to the sender app.js above the **//Setup web server** comment:

```
// Connect client to broker  var topic = 'topic1';  var sendClient =
mqlight.createClient(opts);  sendClient.on('started', function() {
sendClient.send(topic, 'Hello World!');  });  sendClient.start();
```

The remaining step is to intercept web requests and send a message when a request is received.Express uses ‘middleware’ to chain together handlers.The following code inserts a new middleware to publish the request url.This code needs to be added inbetween the var app = express(); and app.listen(appport); lines :

```
//middleware to publish web requests received  app.use(function(req, res, next) {
sendClient.send(topic, 'URL received = ' + req.url);  next();  });
```

This completes the sender app.The entire application **app.js** should now contain the following – Note I’ve added a couple of basic handlers to return content to the requesting browser:

```
var mqlight = require('mqlight');  var express = require('express');  var appport =
process.env.VCAP_APP_PORT || 3000;  //get connection settings for MQLight  if
(process.env.VCAP_SERVICES) {  //running in bluemix  var credentials =
JSON.parse(process.env.VCAP_SERVICES)['mqlight'][0].credentials;  opts = {user:
credentials.username,   password: credentials.password,
service:credentials.connectionLookupURI};  } else {  //running locally  opts =
{service:'amqp://localhost:5672'};  }  // Connect client to broker  var topic =
'topic1';  var sendClient = mqlight.createClient(opts);  sendClient.on('started',
function() {  sendClient.send(topic, 'Hello World!');  });  sendClient.start();
//Setup web server  var app = express();  //middleware to publish web requests
received  app.use(function(req, res, next) {  sendClient.send(topic, 'URL received =
' + req.url);  next();  });  // Add a handler for /  app.get('/', function(req, res,
```

```
next) {  res.send('Hello World');  });  // Add a handles for /help  app.get('/help',
function(req, res, next) {  res.send('No help here!');  });  app.listen(appport);
```

The receiver application will subscribe to the ‘topic1’ topic and log to the console any messages received.The code to be added to the receiver app.js file is:

```
// Connect client to broker  var topic = 'topic1';  var recvClient =
mqlight.createClient(opts);  recvClient.on('started', function() {
recvClient.subscribe(topic);  recvClient.on('message', function(data, delivery) {
console.log(data);  });  });  recvClient.start();
```

To complete receiver app.js file should now contain:

```
var mqlight = require('mqlight');  var express = require('express');  var appport =
process.env.VCAP_APP_PORT || 3000;  //get connection settings for MQLight  if
(process.env.VCAP_SERVICES) {  //running in bluemix  var credentials =
JSON.parse(process.env.VCAP_SERVICES)['mqlight'][0].credentials;  opts =
{user:credentials.username ,   password: credentials.password,
service:credentials.connectionLookupURI};  } else {  //running locally  opts =
{service:'amqp://localhost:5672'};  }  // Connect client to broker  var topic =
'topic1';  var recvClient = mqlight.createClient(opts);  recvClient.on('started',
function() {  recvClient.subscribe(topic);  recvClient.on('message', function(data,
delivery) {  console.log(data);  });  });  recvClient.start();  //Setup web server
var app = express();  app.listen(appport);
```

The last task to complete is to ensure the MQLight service is bound to the application at deploy time.To achieve this the manifest.yml file for both the sender and receiver needs to be modified to include the service.The complete manifest file for the receiver should now contain:

```
---  applications:   - name: MQLrec  host: MQLrec-${random-word}  path: .
instances: 1  memory: 128M  services:   - MQLight
```

and the sender manifest.yml file should contain:

```
---  applications:   - name: MQLsnd  host: MQLsnd-${random-word}  path: .
instances: 1  memory: 128M  services:   - MQLight
```

You can now deploy both the receiver and the sender applications – you need 256MB memory, so ensure you have sufficient resources, if not delete applications from previous exercises.

If using projects, the manifest files can be created while deploying and the service can be bound through the deploy dialog box..

To test the applications the command line interface can be used to tail the log of the receiver application.In a command line window enter the following:

```
cf logs MQLrec
```

Launch the sender application in a browser

In the console window you should see the message output:

cf logs MQLrec

Connected, tailing logs for app MQLrec in org viny.gupta@avnet.com / space vinay as vinay.gupta@avnet.com...

OUT URL received = /

You can also try adding different ending to the URL in the browser, such as /help – this should display ‘no help here!’ in the browser window and also output

OUT URL received = /help

to the console where the log is being tailed.

This concludes the exercise. You should now be familiar how to add a service to an application, how to parse the connection information for the service.The service documentation should explain how to use the service.