Cloud Developer Certification Preparation

Exercise 4.3:
Messaging in the cloud

Exercise 4.3: Prerequisites

Sign up for a 30-day free trial IBM Bluemix account if you don't already have one.

You also need the following software:

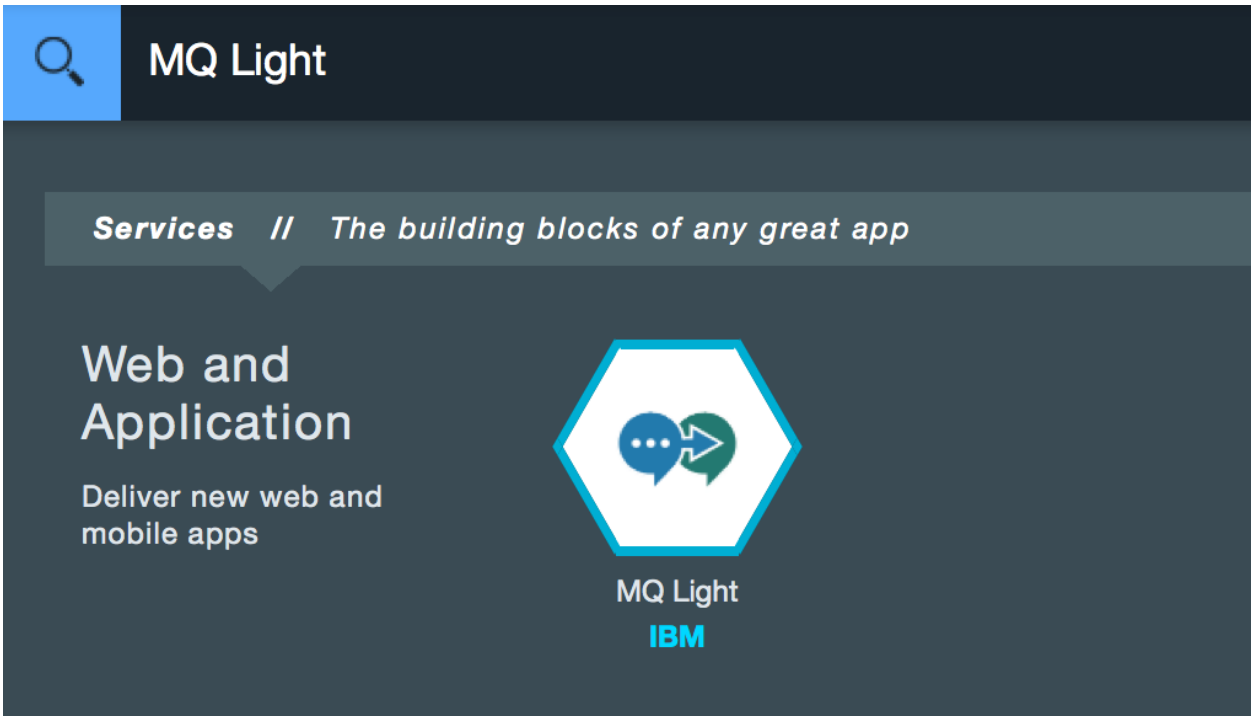
- A web browser supported by Bluemix:
    - Chrome: the latest version for your operating system
    - Firefox: the latest version for your operating system and ESR 31 or ESR 38
    - Internet Explorer: version 10 or 11
    - Safari: the latest version for the Mac
- DevOps project access:
    - https://hub.jazz.net/project/ecosysdevcnc/certification-nodejs-mqlight/overview

Exercise 4.3.1: Creating an MQ Light service instance

Complete the following steps to add a MQ Light service to your application:

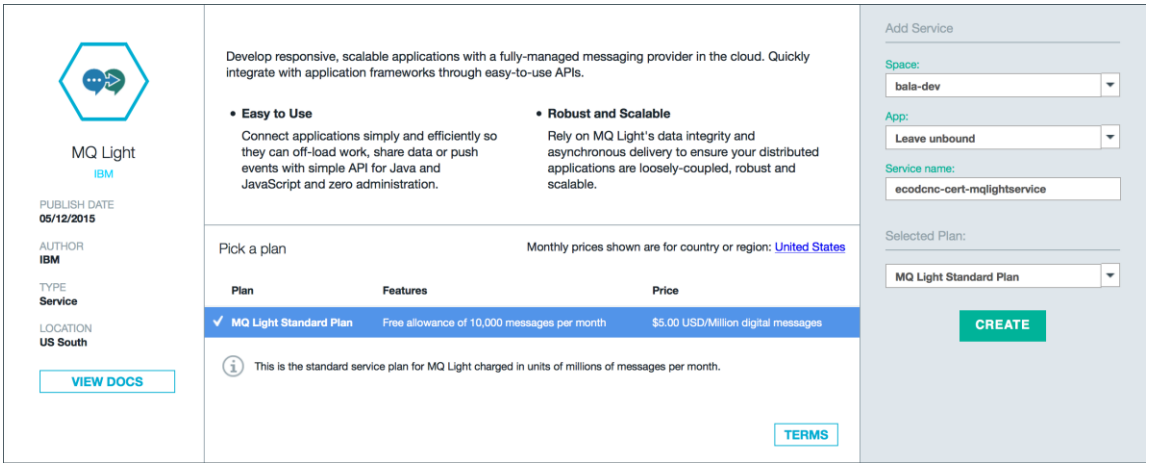
1. In the Bluemix Dashboard, click ADD A SERVICE.

2. In the search field, enter MQ Light. Then, click MQ Light



3. Select your application in the **App** field and then click **CREATE**. You can also create the same MQ Light Service instance by using the command line that can be used by the apps deployed on to Bluemix, for example, enter:

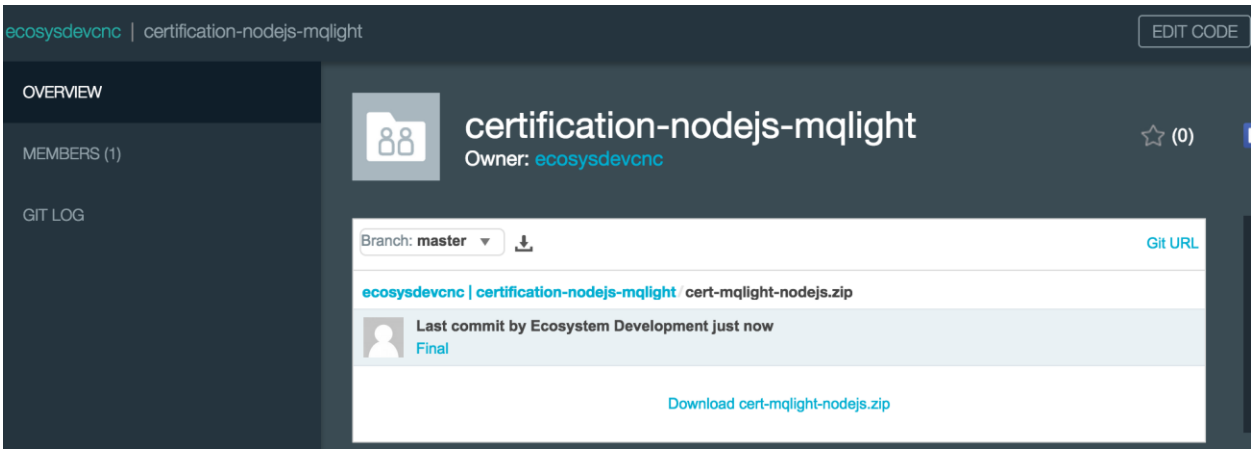
```
cf cs ecodcnc-cert-mqlightservice
```



Exercise 4.3.2: Deploying the sample application to Bluemix

1. Download the code (cert-mqlight-nodejs.zip) from the following Jazz hub website:

   https://hub.jazz.net/project/ecosysdevcnc/certification-nodejs-mqlight/overview



2. Push the apps by navigating to the root directory where you downloaded the apps and running this command:

   ```
   cf push
   ```

First, this command uploads the app files and then sorts out all of the setup for you by taking information from the package.json to download the Node dependencies. This command also binds the service that you created to both the apps that were pushed and starts them.

To do all of these tasks, the `push` command uses the manifest.yml that is in the root directory to facilitate the deployment. This file is optional, but avoids having to specify all of the arguments each time that you run the command. Use the command `cf push -h` if you want to specify these manually.

```
applications:
- name: ecodcnc-cert-mqlight-backend
  memory: 128M
  host: ecodcnc-cert-backend-${random-word}
  path: worker_backend
  command: node app.js
  no-route: true
  services:
  - ecodcnc-cert-mqlightservice
- name: ecodcnc-cert-mqlight-frontend
  memory: 128M
  host: ecodcnc-cert-frontend-${random-word}
  path: worker_frontend
  command: node app.js
  services:
  - ecodcnc-cert-mqlightservice
```

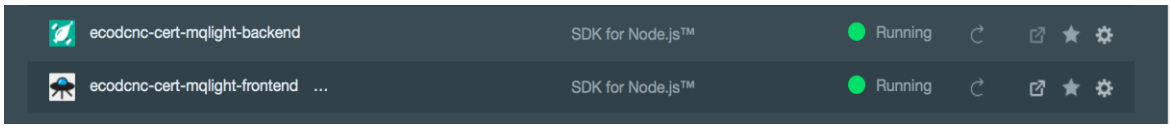
- `name` is the name of your application in Bluemix.
- `disk` is how much disk space your application has.
- `command` is the command that is run to start your app after it is configured in Bluemix. In this case, it is the same command that you used to start it locally.
- `path` is the path to the app on your local machine.
- `memory` is how much RAM you assign the app.
- `instances` indicate how many instances of the app that Bluemix creates, which is relevant only to the back end worker.
- `no-route` indicates that the back end does not need a route (URL) created for it.
- `host` indicates the route (URL) to create for the front end, that is, where the web app is available. Note that `${random-word}` generates a random word that should lower the chances of trying to create an already taken route.
- `services` lists the services that you want to bind the app to during creation. In this case, it is the MQ Light service that you created in the previous step.

3. When it's done, run the `cf apps` command or use the Bluemix dashboard to see that the apps are running.

    - Use the `cf apps` to see the status of applications.

```
ecodcnc-cert-mqlight-backend     started     1/1     128M     1G
ecodcnc-cert-mqlight-frontend    started     1/1     128M     1G     ecodcnc-cert-frontend-afflictive-csch.mybluemix.net
```

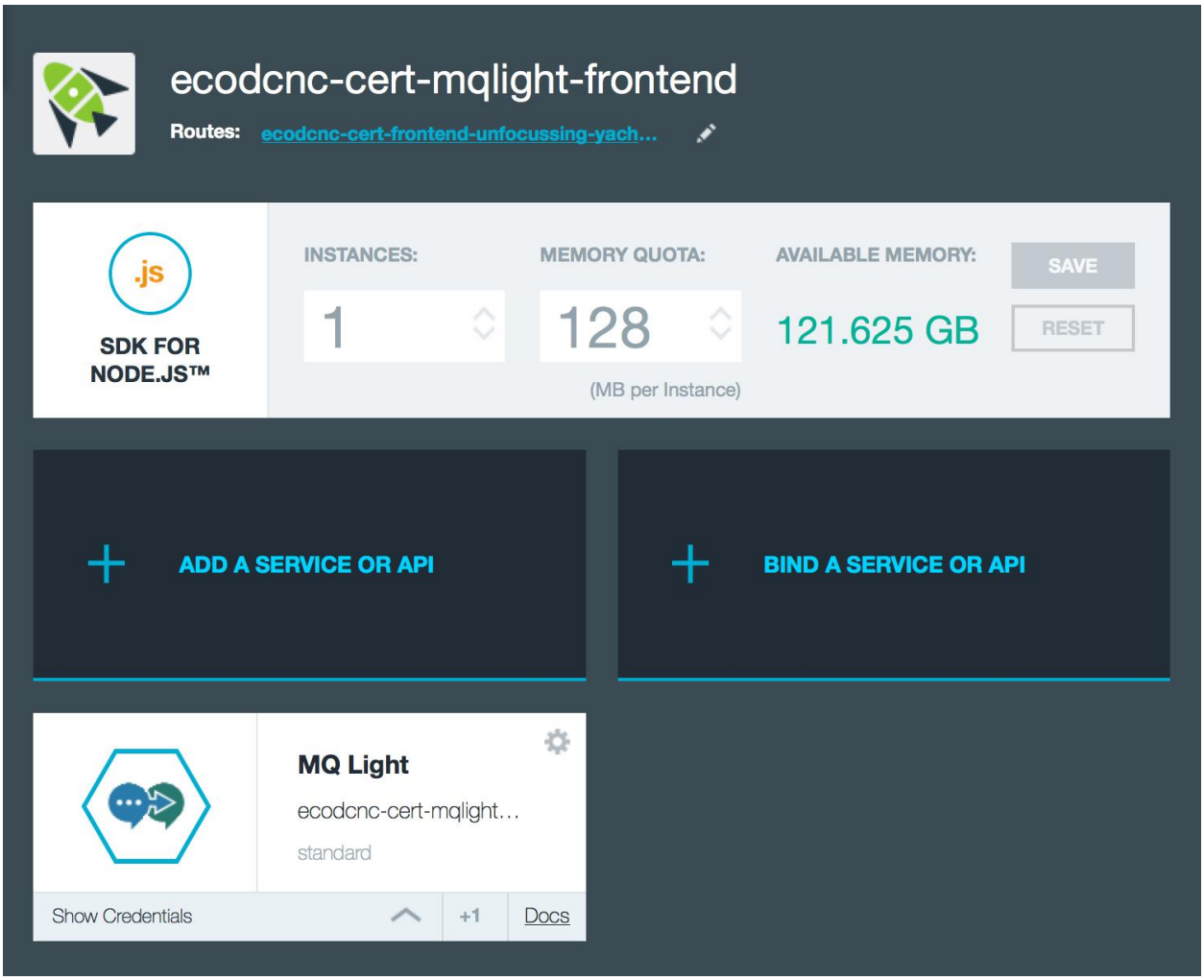
    - Use the Bluemix dashboard to check the status of the applications.



Exercise 4.3.3: Touring the MQ Light dashboard in Bluemix

To view to the MQ Light dashboard:

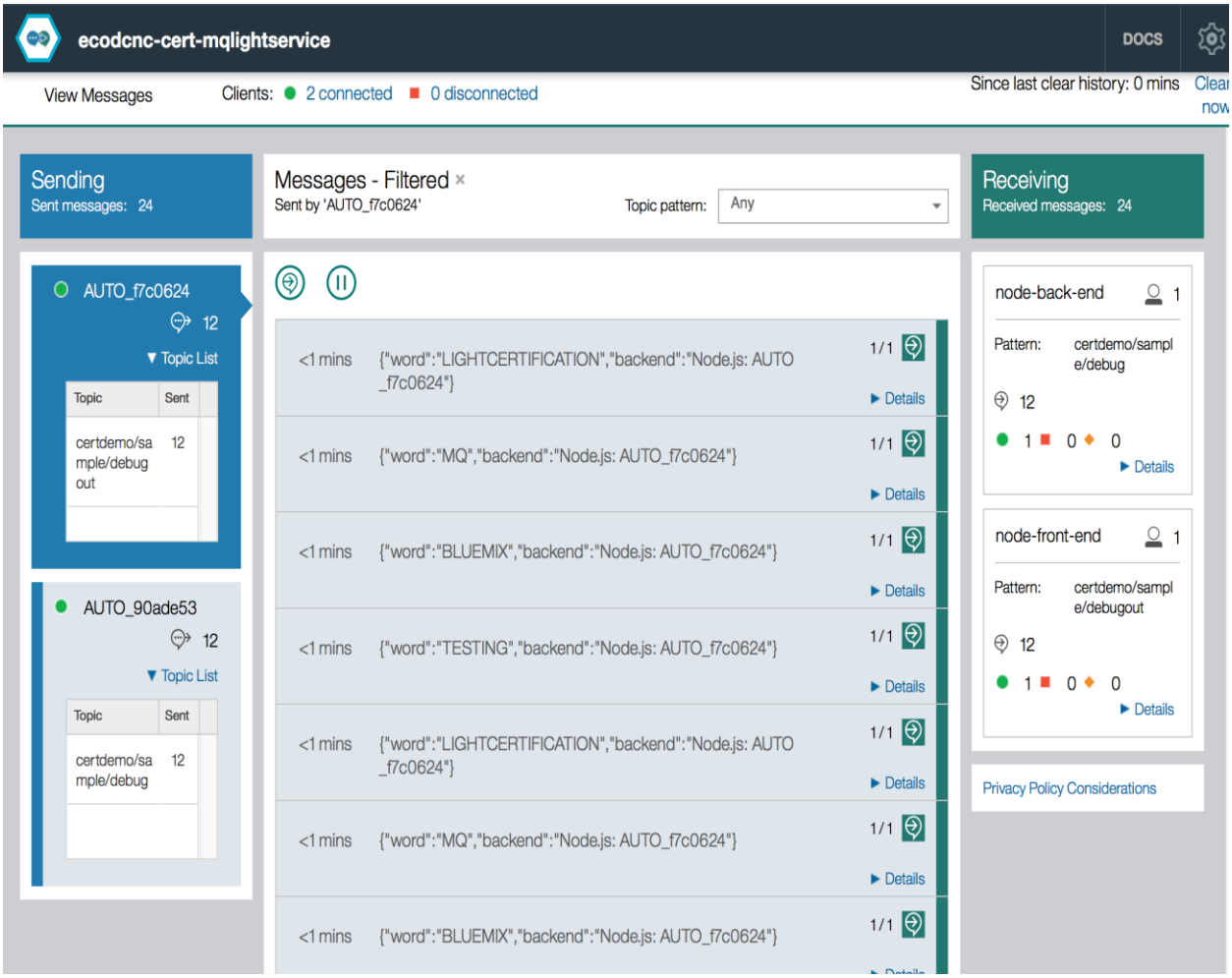
1. For your app running on Bluemix, log in to the Bluemix MQ Light Service UI and from the dashboard, click your instance of the MQ Light service.



You should be able to see senders appearing on the left side, and receivers on the right side and any messages flowing in the middle. For example, in the sample app, after you click **Submit work**, you should see something similar to the screen capture below.

2. Click **Topic List** for senders and **Details** for receivers to get more information about the connected applications.

3. Click **Details** in any of the messages to get some more information, such as the payload, who sent it, to where, and whether anyone is waiting for it. This information can help you troubleshoot messaging applications because you can see what your messages are doing.



For More information about the MQ Light UI, see [https://developer.ibm.com/messaging/mq-light/docs/ui-reference/.](https://developer.ibm.com/messaging/mq-light/docs/ui-reference/)

Exercise 4.3.4: Code walkthrough

By reviewing the ecodcnc_cert-mqlight-frontend_node/app.js code, you can understand how the MQ Light service works. You check whether you're running in Bluemix by checking for the presence of the process.env.VCAP_SERVICES variable. If you find it, you can get the necessary details from Bluemix. Otherwise, you can run it locally and use the default local settings.

```
if (process.env.VCAP_SERVICES) {
  var services = JSON.parse(process.env.VCAP_SERVICES);
  console.log( 'Running BlueMix');
  if (services[ 'mqlight' ] == null) {
    throw 'Error - Check that app is bound to service';
  }
  mqlightService = services['mqlight'][0];
  opts.service = mqlightService.credentials.connectionLookupURI;
  opts.user = mqlightService.credentials.username;
  opts.password = mqlightService.credentials.password;
} else {
  opts.service = 'amqp://localhost:5672';
}
```

You can then start the MQ Light client using these options:

```
var mqlightClient = mqlight.createClient(opts, function(err) {
  ...
```

Here, you can see the app subscribing and specifying the processMessage as the callback that is called when a message arrives:

```
mqlightClient.on('message', processMessage);
  mqlightClient.subscribe(SUBSCRIBE_TOPIC, SHARE_ID,
    {credit : 1,
      autoConfirm : false,
      qos : 1}, function(err) {
        if (err) console.err("Failed to subscribe: " + err);
        else {
          console.log("Subscribed");
          mqlightSubInitialised = true;
        }
      });
```

You are subscribing with a credit of 1 and autoConfirm set to false meaning that you deal with one message at a time until you confirm that you have handled it. The rest of the code is self-explanatory.