How to use

This project depends on and was tested with the following Node.js libraries:

- fivebeans version 1.3.1
- js-yaml version 3.3.1
- mongodb version 2.0.39
- yargs version 3.15.0

Before you run anything, configure the connection to beanstalk and mongo DB in the configuration file (**config.yml**) All parameters for job management (such as the maximum number a job is allowed to run before it fails or the delay that it incurs in the tube as a result) can be configured in **config.yml**. For all parameters, please, see **config.yml**.

You can do the following with this project:

- To start the consumer worker, run **worker.js** in the main directory. To run worker.js in multiple instances of Node.js, specify an ID for each instance as follows:

  ```
  node worker.js --id=worker2
  ```

  This allows the tube to know who is watching it. If running worker from different computers, the ID of each worker can be the IP of the computer.
- To put an initial seed into the tube, run **seeder.js** in the main directory.
- To list the status of the tube, run **list_tube_info.js** in the main directory.

If running any of the above modules from a directory other than the main directory, pass the configuration file to the module as follows:

```
node /dev/quoter/seeder.js --config=/dev/quoter/config.yml
```

Design

Introduction

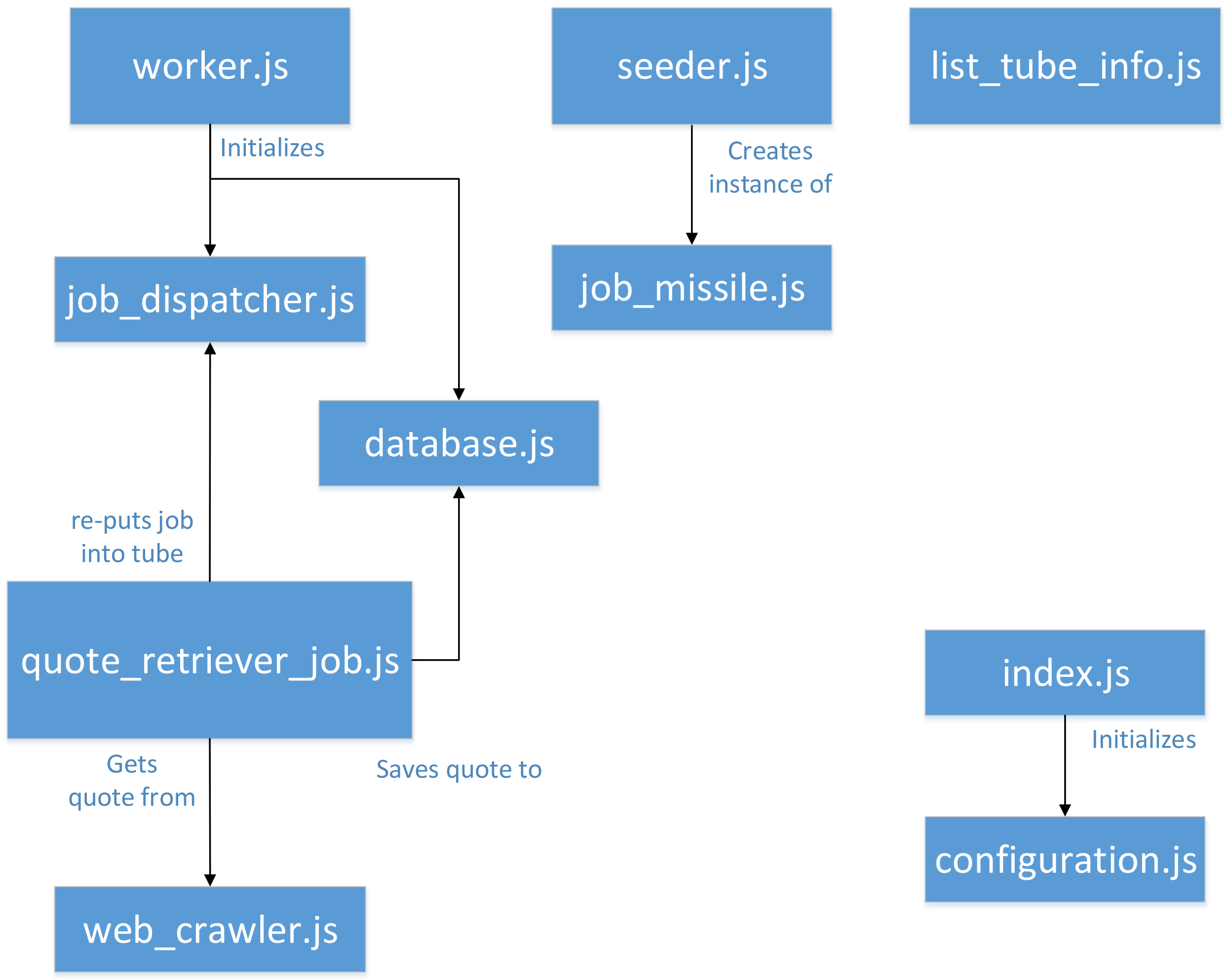
This is my first JavaScript project, using MongoDb for the first time, and using beanstalkd for the first time, and Node.js. I guess, I have a few firsts here. So, it was exiting.

Here are a few design considerations:

1. Fivebeans lib performs asynchronous execution of **asynchronous** jobs (emphasis on asynchronous). So, Fivebeans can be executing many jobs at the same time within the same Node.js instance. And our quote job is asynchronous by nature since it issues requests to the web (which are asynchronous themselves).
2. Database connection pool keeps 5 active connections to the database.
3. Javascript closures are used to avoid sharing data between asynchronous jobs executing in parallel.
4. I know Java object oriented design so I tried to follow the Java object oriented design patters. I recognize that there is an entirely different set of unique design patters for Javascript. I tried a few and I am learning …
5. Plain TCP is used for transport to beanstalkd. I couldn't get the HTTP POST to Aftership working. See section at end. Maybe I need help here.
6. Quotes are retrieved from http://themoneyconverter.com using an HTTP GET. I do not have a key for XE.com
7. Error handling is implemented for the web component that retrieves the quotes and for the database component partially. Losing connection to beanstalk will yield undefined behavior.
8. Testing: I performed manual testing with 3 jobs in one Node.js and with 2 jobs in two Node.js instances. I could write unit tests as a next step but it will be another first for me in Javascript.
9. I left some @todo-s that are still outstanding. Following up on these will improve design and error handling.

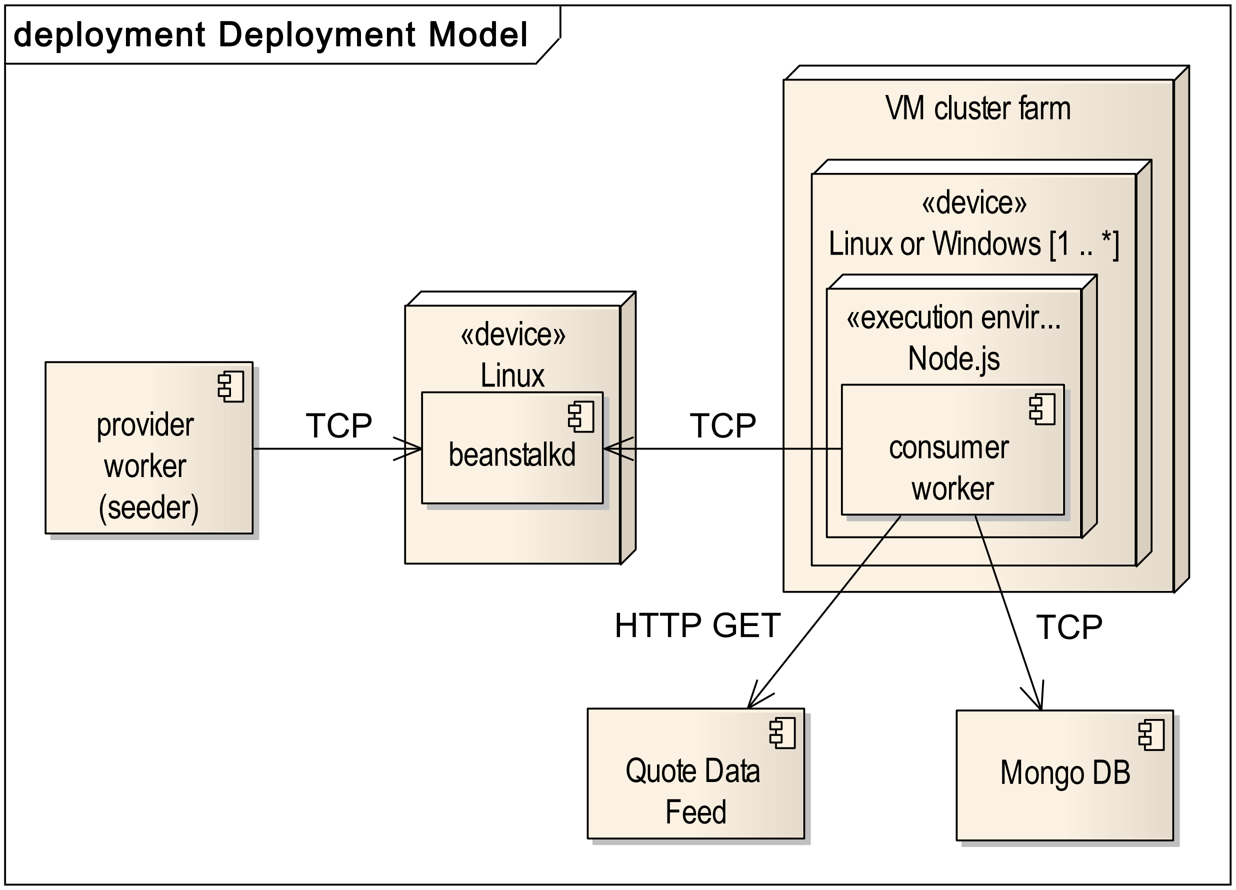
Component Diagram

Below is a package diagram. Worker.js seeder.js and list_tube_info.js are the main files that initiate execution of the relevant module. Worker.js initializes connections to the database and to beanstalkd tube. Quote_retriever_job.js manages the state of the job, calling web_crawler.js to get the quote, calling database.js to save the quote and calling job_dispatcher.js to save the job back to the tube. Seeder.js puts creates job_missile and puts into the tube. Index.js and configuration.js are used by all the rest of the modules.



Deployment Diagram

Below is a specification-level deployment diagram. It shows that the consumer worker can run on multiple environments within a cluster.



Problem connecting to Aftership API

I could not receive an OK message when putting jobs into Aftership's beanstalkd. I used the code snippet below and I receive the following response. What should I change so I can use Aftership's beanstalkd?

```
>echo output from Aftership
>node aftership.beanstalk.js
{"meta":{"code":200},"data":{"host":"challenge.aftership.net","port":11300}}
{"meta":{"code":200},"data":{"host":"challenge.aftership.net","port":11300}}
{"meta":{"code":200},"data":{"host":"challenge.aftership.net","port":11300}}
```

```
// code snippet from aftership.beanstalk.js
var http = require('http');
function performRequest(command, data) {

       var dataString;
       if (data)
              dataString = command + data.length  + '\r\n' + JSON.stringify(data);
       else
              dataString = command + '\r\n';

       var options = {
              hostname: 'challenge.aftership.net'
              , port: 9578 //11300
              , path: '/v1/beanstalkd'
              , method: 'POST'
              , headers: {
                     'Content-Type': 'application/json'
                     , 'aftership-api-key': 'a6403a2b-af21-47c5-aab5-a2420d20bbec'
                     , 'Content-Length': dataString.length
              }
       };
       var req = http.request(options, function(res) {

              res.setEncoding('utf-8');
              var responseString = '';

              res.on('data', function(data) {
                     responseString += data;
              });
              res.on('end', function() {
                     console.log(responseString);
              });
       });
       req.write(dataString);
       req.end();
}

performRequest('use ivanrachev');
performRequest('put 1000 0 1 ', {"from": "HKD", "to": "USD"});
performRequest('list-tubes');
```