# Bluemix Hands-On Workshop

## Section 6 - Services

Section 6 - Services

**Version:** 15
**Last modification date:** 8-Jul-15
**Owner:** IBM Ecosystem Development

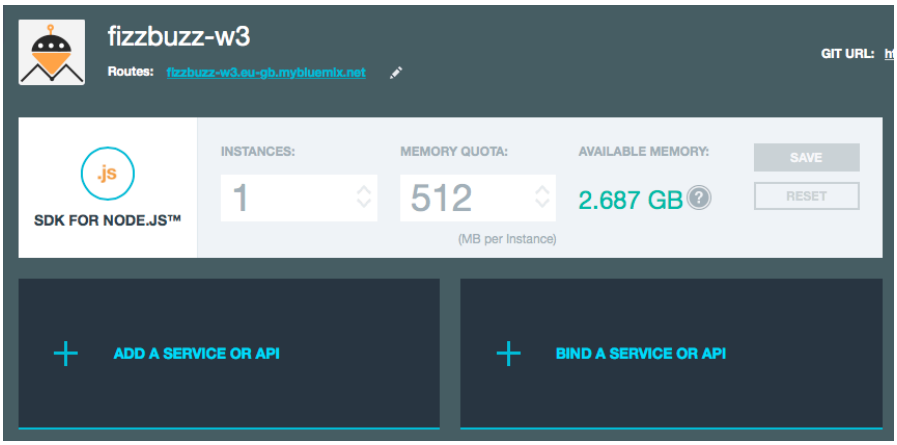
## Table of Contents

# Exercise 6.a – Adding a service to an application

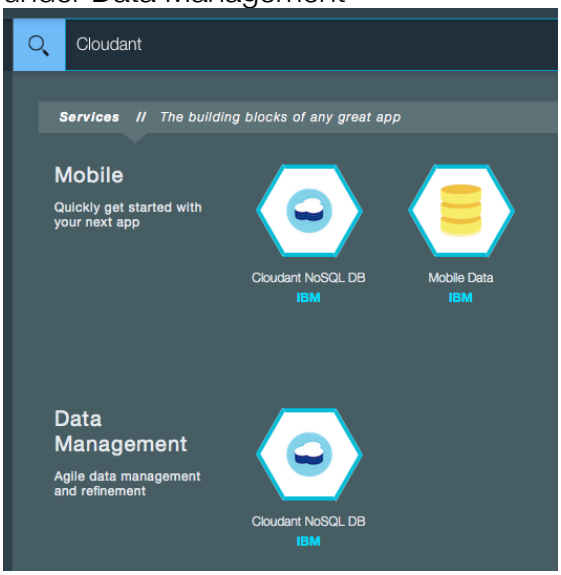
We are going to extend the FizzBuzz implementation to cache results to a NoSQL database. When a range is requested the database will be searched to see of this request has been previously saved. If it has the FizzBuzz result will be returned from the database, if the range has not previously been saved it will be calculated, saved to the database for future requests then the results will be sent to the requesting user.

For this lab the Cloudant database service from Bluemix will be used, so we need to add an instance to the application running on Bluemix.

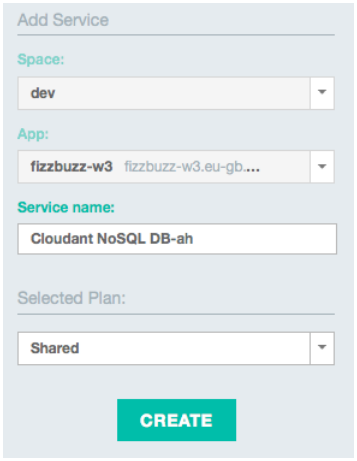
1. Create the Cloudant service for the application
   - In the Bluemix Web UI select your application from the dashboard
   - From the overview page of your Bluemix application select 'ADD A SERVICE OR API'



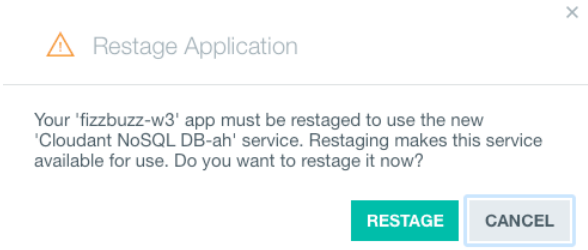

   - Enter Cloudant into the search bar then select the Cloudant NoSQL DB under Data Management




   - The default values should be OK, but you may choose to provide your own service name to be able to better associate the service name with your application

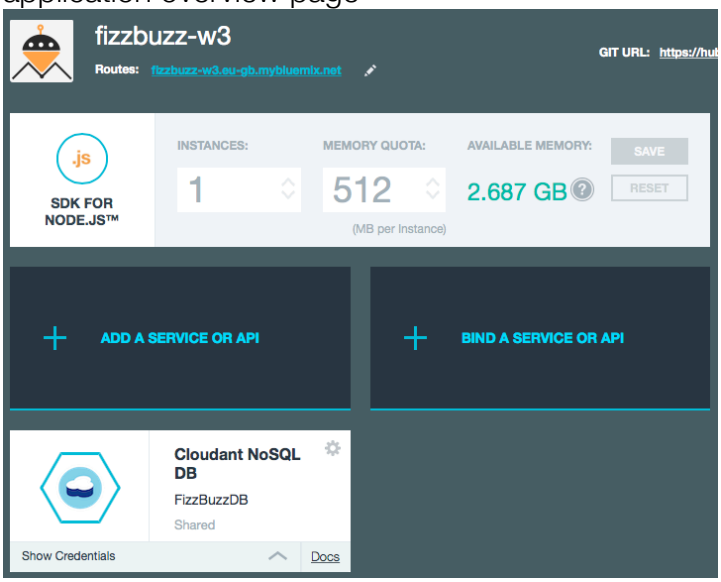



- select CREATE to create and bind the database to your application
- you will be asked to restage the application to make it available to the application. Select RESTAGE

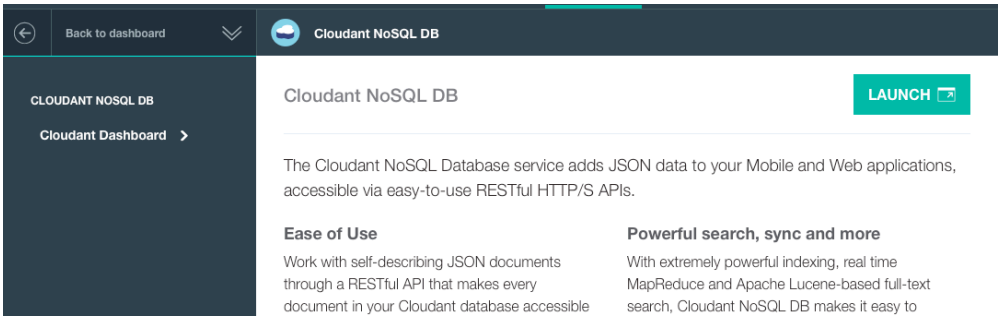



- Wait until the application has finished restaging and is running again

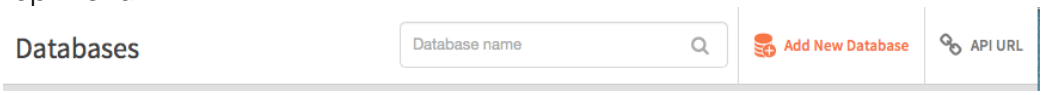
2. Configure the database for the FizzBuzz application
   - To launch the Cloudant console click on the Cloudant service from the application overview page



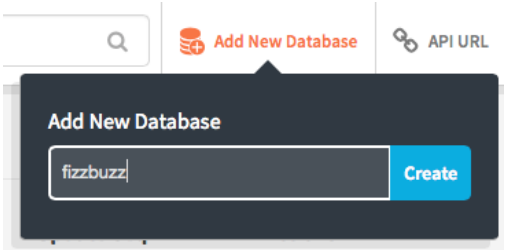

   - then select the LAUNCH button from top right of Cloudant Dashboard




- Once the Cloudant console has loaded select Add New Database from the top menu

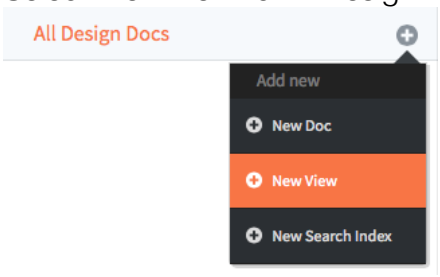



- enter fizzbuzz as the database name

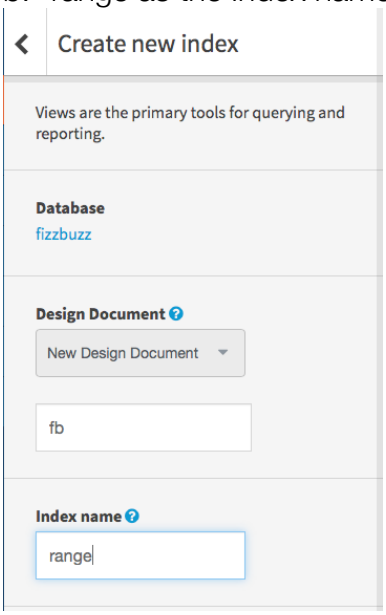



- then press Create

3. Create a view to be able to search for documents with a given range [from, to]
   - Select the + next to All Design Docs




   - In the Create new index panel enter:
     a. fb as the Design Document name
     b. range as the Index name



- change the Map function to the following:

```
function(doc) {
  emit([doc.from,doc.to], null);
}
```


- select Save & Build Index

The database is now ready to receive documents

To use Cloudant within node there are a number of options, from making direct REST API calls against the Cloudant API to using one of the libraries. In this lab we will use the Cloudant library

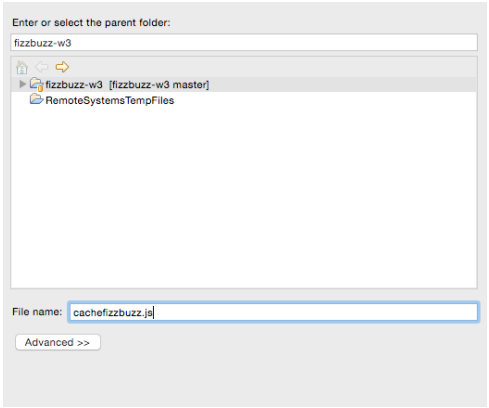
4. Add the Cloudant library to the Eclipse project
   - In Eclipse in the Terminals view (Ctrl+C is Mocha is still running)
   - Enter the following:

     ```
     npm install cloudant@1.0.0 --save --save-exact
     ```

     - This ensures that version 1.0.0 is used and is entered in the package.json file to use exactly 1.0.0, not 1.0.0 or higher.

For this lab we will create a new route in the server so you have access to the caching and non-caching versions of the API.

5. Create the files needed to start implementing the caching version of the API
   - Select the project name in the Project Explorer view then right-click -> New -> File. Name the file cachefizzbuzz.js



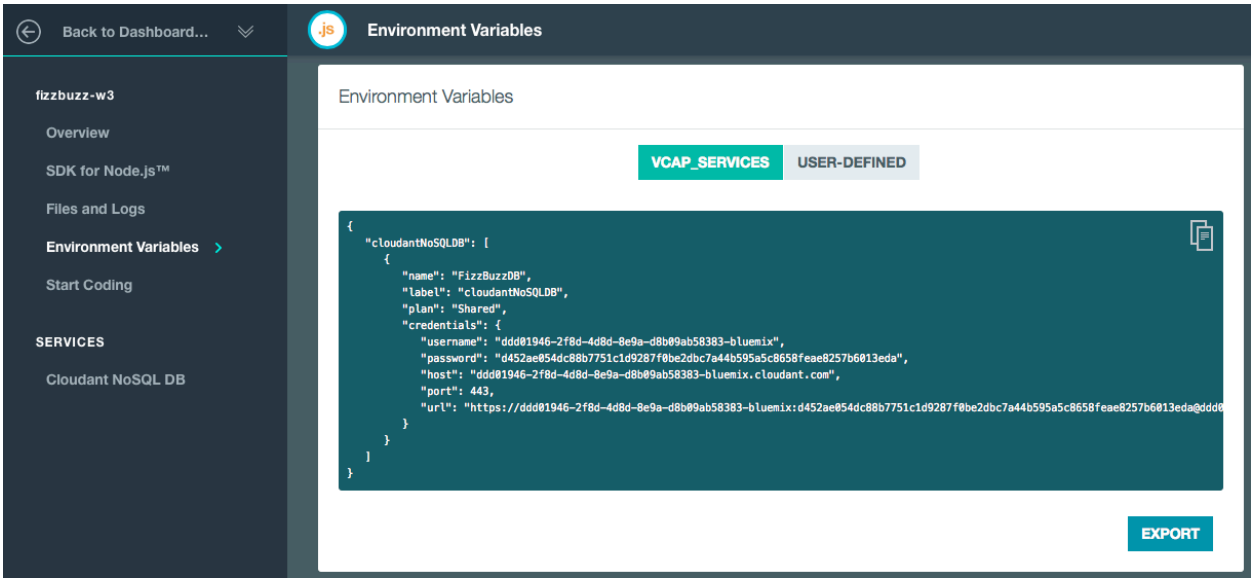
- Enter the following code in the file then save the file

```
var FizzBuzz = require("./fizzbuzz");

var CacheFizzBuzz = function(dbURL) {
  this._dbURL = dbURL;
  this._Cloudant = require("cloudant")(dbURL);
  this._fizzbuzz = new FizzBuzz();
};

module.exports = CacheFizzBuzz;
```

- Select the test directory in the Project Explorer view. right-click -> File -> New. Name the file cachefizzbuzz.test.js
- Enter the following content then save the file:

```
var sinon = require("sinon");
var CacheFizzBuzz = require("../cachefizzbuzz.js");
```

To use the Cloudant database within the application the VCAP_SERVICES environment variable needs to be used to get access to the database location and credentials. The value of the environment variable can be seen in the Bluemix UI.




For the cloudant library only the URL is needed.

6. Parse the VCAP_SERVICES and extract the URL. If the application is not running on Bluemix then create a local URL.
   - In server.js add code to parse the VCAP_SERVICES and extract the database URL under the existing code to obtain the server host and port values
   - In server.js add a new variable called CacheFizzBuzz and initialize it with require("./cachfizzbuzz");

- The top of the server.js file should now contain:

```
var express = require("express");
var app = express();
var FizzBuzz = require("./fizzbuzz");
var CacheFizzBuzz = require("./cachefizzbuzz");

var server_port = process.env.VCAP_APP_PORT || 3000;
var server_host = process.env.VCAP_APP_HOST || "localhost";
var dbURL = "";
if (process.env.VCAP_SERVICES) {
        var env = JSON.parse(process.env.VCAP_SERVICES);
        dbURL = env.cloudantNoSQLDB[0].credentials.url + "/fizzbuzz";
} else dbURL = "http://localhost:5984/fizzbuzz";
```

  - Note: This code assumes you called your database fizzbuzz in step 2.
- The last update to server.js is to add the new route to the server. After the code defining the /fizzbuzz_range route add the following code

```
app.get("/cache_fizzbuzz_range/:from/:to", function (req, res) {
  var cachefizzbuzz = new CacheFizzBuzz(dbURL);
  var from = req.params.from;
  var to = req.params.to;

  cachefizzbuzz.fizzBuzzRange(from, to, function(data) {
          res.send(data);
  });
});
```

  - Note: as a database call is used in the implementation of this API the code needs to use a callback function to make the res.send(data) call as the database call is asynchronous.
- Save the server.js file

To start implementing the new API the lab will not follow a strict test driven approach due to time constraints. It will focus on the requirements to test code invoking a remote service.

The application needs to perform 2 actions against the database:

- Search for a results for a given range [x..y]
- Save the result for a calculated range [x..y]

*Calls to external services need to use asynchronous function calls in JavaScript to avoid blocking the user thread.*

*There are a number of ways to handle asynchronous calls in JavaScript. Passing an anonymous function as a parameter is a common approach, however this can be problematic to test. Having deeply nested stack of callbacks can also make code difficult to understand and maintain. To overcome this problem one approach is to create functions to handle callbacks outside the asynchronous calls1. This way they can be tested individually.*

```
asyncCall(p1, p2, function(err, data) {
   <implementation of callback function>
});
```

*becomes:*

```
function cb(err, data) {
  <implementation of callback function>
}
….
asyncCall(p1, p2, cb);
```

*With this implementation the callback function 'cb' can be unit tested without the 'aysnCall' being invoked. If 'asynCall is a call to a remote server, then we can now test how our code handles the response without any calls to a remote server.*

*Note: JavaScript Promises provide a better way to handle asynchronous behavior, but to keep the code easier to follow for non-JavaScript developers they are not used in this exercise.*

The code to implement the cache functionality is:

```
CacheFizzBuzz.prototype.fizzBuzzRange = function(start, end, callback) {
  var self = this;
  var parms = {
      key : [ start, end ],
      include_docs : true
  };

  self._Cloudant.view('fb', 'range', parms, function(err, body) {
    self._processDBResult(err, body, start, end, callback);
  });
};

CacheFizzBuzz.prototype._dbStoreCalculatedResult = function(data) {
  this._Cloudant.insert(data);
};

CacheFizzBuzz.prototype._processDBResult = function(err, body, start, end,
    callback) {
  var fromDB = false;
  var data = {};
  if ((!err) && (body.rows.length > 0)) {
    delete body.rows[0].doc._id;
    delete body.rows[0].doc._rev;
    data = body.rows[0].doc;
  } else {
    data = {
        "from" : start,
        "to" : end,
        "result" : this._fizzbuzz.convertRangeToFizzBuzz(start, end)
```

```
        };
        if (!err) {
          this._dbStoreCalculatedResult(data);
        }
      }
      callback(data);
    };
```

The above code should be added above the line of code starting **`modules.export`** in file cachefizzbuzz.js.

## Implement the tests for the cachefizzbuzz code (optional)

Testing the above code requires that we isolate the calls to the database and ensure they are note executed. To do this mocks and stubs from sinon are used.

To test the dbStoreCalculatedResult() function we only want to check that we call the Cloudant function with the correct parameters. Unit testing should not check that we correctly create a document in the database.
When we test the function, the call to coudant.insert() should not be made. Sinon provides a stub and mock functionality to replace the call but allow test to be made to verify the call will be correctly made when running in production mode.

- Add the following code to cachefizzbuzz.test.js in the test folder:

```
var testResult1 = {
    "from" : "1",
    "to" : "20",
    "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                 "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz" ]
};

describe("CacheFizzbuzz", function() {
      var f = new CacheFizzBuzz("http://user:password@localhost/fizzbuzz");

      describe("dbStoreCalculatedResult()", function() {
        it("calls Cloudant to store the doc", function() {

          var mock = sinon.mock(f._Cloudant);
          mock.expects("insert").withArgs(testResult1).once();

          f._dbStoreCalculatedResult(testResult1);

          mock.verify();
          mock.restore();
        });
      });

        // remaining tests go above here
 });
```

This test uses the mock functionality of sinon. The _Cloudant object is mocked, so all calls will be captured by the mock. We then setup an expectation that the insert function should be called once with a given set of fizzbuzz results. Once the mock and expectation have been setup the function under test is called, then we ask the mock to verify that all expectations have been met. At the end of the test we restore the _Cloudant object to remove the mock.

- Add the following code to the cachefizzbuzz.test.js file above the line 'remaining tests go above here':

```
describe("fizzBuzzRange()", function() {
            var cbFunction = function(data) {
            };

            it("calls Cloudant to get record from the fb/range view in DB",
                function() {
                  var stub = sinon.stub(f._Cloudant, "view");

                  f.fizzBuzzRange("1", "20", cbFunction);

                  expect(stub.withArgs('fb', 'range', {
                    include_docs : true, key : [ "1", "20" ]
                  }, sinon.match.any).calledOnce).to.be
                  .eql(true,
                  "Expected cloudant.view to be called only once with correct
parameters");

                  f._Cloudant.view.restore();
                });
          });
```

The above code tests the fizzBuzzRange() function. This function asks the database for any stored results for the range specified by the start and end input parameters. The returned results are then passed to the _processDBResult() function. To test the function we provide a fake callback function 'cbFunction' and use the stub feature of sinon to ensure no call to Cloudant is made. Unlike the mock, the expectations are tested after the function under test is called. The anonymous function passed as the callback function to the cloudant view function call can be ignored in the test by using 'sinon.match.any'.

The final part to test is the function to process the return from the Cloudant.view call. This function should look at the results returned. If there was a document found matching the input key, then the result should be returned. If not the result needs to be calculated using the functionality implemented in the previous exercise and the result should be stored in the database before returning the result.

- To create the tests there needs to be additional test data, so under the definition of test1 add the following code:

```
var testResult2 = {
    "from" : "2",
    "to" : "21",
    "result" : ["2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz",
"Fizz"]
};

var DBResult1 = {
    "total_rows" : 7,
    "offset" : 6,
    "rows" : [ {
      "id" : "35523244d141fb56c2e6b8dfa58d7fed",
      "key" : [ "1", "20" ],
      "value" : null,
      "doc" : {
        "_id" : "35523244d141fb56c2e6b8dfa58d7fed",
        "_rev" : "1-a0337a71e877726cb8fda7d6ceeb2bfc",
        "from" : "1",
        "to" : "20",
        "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz",
                     "Buzz", "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz",
                     "19", "Buzz" ]
      }
    } ]
};
```

- There are 3 test scenarios to cover:
  1. Results are found in the database
  2. No results are found in the database
  3. Database returns an error
- Add the following code below the previous test, but above 'remaining tests go above here':

```
describe("processDBResult()", function() {
  var cbFunction = sinon.spy();
  var fizzBuzzSpy = null;

  beforeEach( function() {
    fizzBuzzSpy = sinon.spy(f._fizzbuzz, "convertRangeToFizzBuzz");
  });

  afterEach( function() {
    cbFunction.reset();
    f._fizzbuzz.convertRangeToFizzBuzz.restore();
  });
```

```
            it("processes the results from Cloudant with a valid result set",
                function() {
                  var storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult to parse DB return");
                  expect(fizzBuzzSpy).callCount(0);
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates the results when no results found then saves to DB",
                function() {
                  storeResultsStub = sinon.stub(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult2, "2", "21", cbFunction);

                  expect(cbFunction.withArgs(testResult2).calledOnce).to.be
                  .eql(true,"Expected calculated result when no data from DB");
                  expect(fizzBuzzSpy.withArgs("2", "21").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate
results");
                  expect(storeResultsStub.withArgs(testResult2).calledOnce).to.be
                  .eql(true, "Expect calculated restuls to be stored in DB");

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates results when DB error,but doesn't store results",
                function() {
                  storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(true, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult create empty result with DB
error");
                  expect(fizzBuzzSpy.withArgs("1", "20").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate
results");
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });
        });
```

- To avoid repetition in tests the beforeEach and afterEach hook functions are used to set up the spies and stubs used in each test.

All the tests should pass.

This should give you a good idea of how a unit test can be used when services are used. Unit Testing is a large subject, so only a brief example has been possible in this course.

# Exercise 6.b – Creating a user-provided service

Bluemix allows you to create a service local to your organization. The service needs to be running, not necessarily on Bluemix, but must be reachable from Bluemix. The service definition serves as a link between the running service and your Bluemix application.

To create a service use the Command Line Interface:

```
cf cups testService -p "host, port, user, password"
```

You will then be prompted for the values of the parameters. The values you enter will be stored with the service definition and passed to the applications binding to your service

```
cf cups testService -p "host, port, user, password"

host> myhost.ibm.com

port> 12345

user> user1

password> passw0rd
Creating user provided service testService in org binnes@uk.ibm.com / space test as
binnes@uk.ibm.com...
```

It is also possible to specify the values in the command rather than being prompted:

```
cf cups testService -p '{"host":"myhost.ibm.com", "port":"12345", "user":"user1",
"password":"passw0rd"}'
```

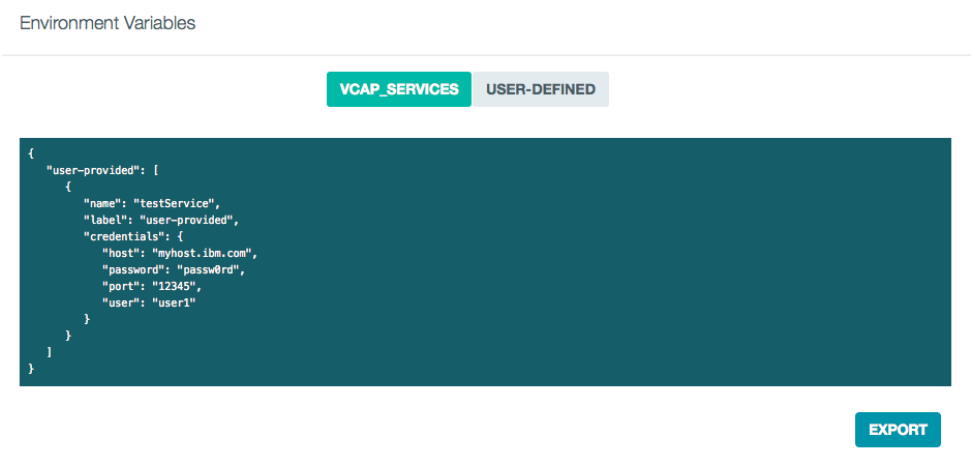
Create your own user-defined service and add parameters for the service using the –p option. You can provide the values on the command line or be prompted for the parameters.

You can list the services and see the service is now available using **`cf s`**:

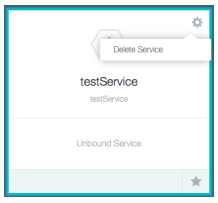
```
cf s
Getting services in org binnes@uk.ibm.com / space test as binnes@uk.ibm.com...
OK

name                                service           plan      bound apps
BIJavaCloudantTest:cloudantNoSQLDB  cloudantNoSQLDB   Shared    BIJavaCloudantDBApp
BINodeWebStarter:DataCache          DataCache         free      BINodeWebStarter
MQLight                             mqlight           Default
MQLight-yh                          mqlight           Default   BINodeWebStarter,
MQLrec, MQLsnd
testService                         user-provided
```

If you need to update the values for the service use **`cf uups`**.

You can bind the service to an application like any other Bluemix service and the application can parse VCAP_SERVICES to access the parameters you provided:




User-provided services can be deleted like any other Bluemix service using the Bluemix Web UI :




or using the command line **`cf ds`** :

```
cf ds testService

Really delete the service testService?> y
Deleting service testService in org binnes@uk.ibm.com / space test as
binnes@uk.ibm.com...
OK
```

# Appendix I - Sample code for the fizzbuzz labs

Fizzbuzz.js:

```
var FizzBuzz = function (){
};

FizzBuzz.prototype.divisibleBy = function(number, divisor) {
        return number % divisor === 0;
};

FizzBuzz.prototype.convertToFizzBuzz = function(number) {
          if (this.divisibleBy(number, 15)) {
            return "FizzBuzz";
          }

          if (this.divisibleBy(number, 3)) {
            return "Fizz";
          }

          if (this.divisibleBy(number, 5)) {
            return "Buzz";
          }

          return number.toString();
};

FizzBuzz.prototype.convertRangeToFizzBuzz = function(start, end) {
          var result = [];
      var from = parseInt(start);
      var to = parseInt(end);

          for (var i = from; i <= to; i++) {
            result.push(this.convertToFizzBuzz(i));
          }

          return result;
};

module.exports = FizzBuzz;
```

cachefizzbuzz.js:

```
var sinon = require("sinon");
//var CacheFizzBuzz = require("../cachefizzbuzzpromise.js");
var CacheFizzBuzz = require("../cachefizzbuzz.js");

var testResult1 = {
    "from" : "1",
    "to" : "20",
    "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                 "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz" ]
};

var testResult2 = {
    "from" : "2",
    "to" : "21",
    "result" : ["2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz",
"Fizz"]
};

var DBResult1 = {
    "total_rows" : 7,
    "offset" : 6,
    "rows" : [ {
      "id" : "35523244d141fb56c2e6b8dfa58d7fed",
      "key" : [ "1", "20" ],
      "value" : null,
      "doc" : {
        "_id" : "35523244d141fb56c2e6b8dfa58d7fed",
        "_rev" : "1-a0337a71e877726cb8fda7d6ceeb2bfc",
        "from" : "1",
        "to" : "20",
        "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz",
```

```
                    "Buzz", "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz",
                    "19", "Buzz" ]
      }
    } ]
};

var DBResult2 = {
    "total_rows" : 7,
    "offset" : 7,
    "rows" : []
};

describe("CacheFizzbuzz", function() {
      var f = new CacheFizzBuzz("http://user:password@localhost/fizzbuzz");

      describe("dbStoreCalculatedResult()", function() {
        it("calls Cloudant to store the doc", function() {

          var mock = sinon.mock(f._Cloudant);
          mock.expects("insert").withArgs(testResult1).once();

          f._dbStoreCalculatedResult(testResult1);

          mock.verify();
          mock.restore();
        });
      });

      describe("fizzBuzzRange()", function() {
            var cbFunction = function(data) {
            };

            it("calls Cloudant to get record from the fb/range view in DB",
                function() {
                  var stub = sinon.stub(f._Cloudant, "view");

                  f.fizzBuzzRange("1", "20", cbFunction);

                  expect(stub.withArgs('fb', 'range', {
                    include_docs : true, key : [ "1", "20" ]
                  }, sinon.match.any).calledOnce).to.be
                  .eql(true,
                  "Expected cloudant.view to be called only once with correct
parameters");

                  f._Cloudant.view.restore();
                });
          });

      describe("processDBResult()", function() {
        var cbFunction = sinon.spy();
        var fizzBuzzSpy = null;

        beforeEach( function() {
          fizzBuzzSpy = sinon.spy(f._fizzbuzz, "convertRangeToFizzBuzz");
        });

        afterEach( function() {
          cbFunction.reset();
          f._fizzbuzz.convertRangeToFizzBuzz.restore();
        });

            it("processes the results from Cloudant with a valid result set",
                function() {
                  var storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult to parse DB return");
                  expect(fizzBuzzSpy).callCount(0);
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates the results when no results found then saves to DB",
```

```
                function() {
                  storeResultsStub = sinon.stub(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult2, "2", "21", cbFunction);

                  expect(cbFunction.withArgs(testResult2).calledOnce).to.be
                  .eql(true,"Expected calculated result when no data from DB");
                  expect(fizzBuzzSpy.withArgs("2", "21").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate
results");
                  expect(storeResultsStub.withArgs(testResult2).calledOnce).to.be
                  .eql(true, "Expect calculated restuls to be stored in DB");

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates results when DB error, but doesn't store results",
                function() {
                  storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(true, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult create empty result with DB
error");
                  expect(fizzBuzzSpy.withArgs("1", "20").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate
results");
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });
          });

      //remaining tests go above here
});
```

server.js:

```
var express = require("express");
var app = express();
var FizzBuzz = require("./fizzbuzz");
var CacheFizzBuzz = require("./cachefizzbuzz");

var server_port = process.env.VCAP_APP_PORT || 3000;
var server_host = process.env.VCAP_APP_HOST || "localhost";
var dbURL = "";
if (process.env.VCAP_SERVICES) {
        var env = JSON.parse(process.env.VCAP_SERVICES);
        dbURL = env.cloudantNoSQLDB[0].credentials.url + "/fizzbuzz";
} else dbURL = "http://localhost:5984/fizzbuzz";

app.get("/fizzbuzz_range/:from/:to", function (req, res) {
  var fizzbuzz = new FizzBuzz();
  var from = req.params.from;
  var to = req.params.to;

  res.send({
    from: from,
    to: to,
    result: fizzbuzz.convertRangeToFizzBuzz(from, to)
  });
});

app.get("/cache_fizzbuzz_range/:from/:to", function (req, res) {
  var cachefizzbuzz = new CacheFizzBuzz(dbURL);
  var from = req.params.from;
  var to = req.params.to;

  cachefizzbuzz.fizzBuzzRange(from, to, function(data) {
          res.send(data);
  });
});

var server = app.listen(server_port, server_host, function () {

  var host = server.address().address;
```

```
  var port = server.address().port;

  console.log("Example app listening at http://%s:%s", host, port);

});
```

test/fizzbuzz.test.js:

```
var sinon = require("sinon");
var FizzBuzz = require("../fizzbuzz.js");

describe("Fizzbuzz", function() {
  var f = new FizzBuzz();

  describe("divisibleBy()", function() {
            it("when divisible", function() {
              expect(f.divisibleBy(3, 3)).to.be.eql(true);
            });

            it("when not divisible", function() {
              expect(f.divisibleBy(3, 2)).to.be.eql(false);
            });
  });

  describe("convertToFizzBuzz()", function() {
            it("when divisible by 3", function() {
              expect(f.convertToFizzBuzz(3)).to.be.equal("Fizz");
              expect(f.convertToFizzBuzz(6)).to.be.equal("Fizz");
            });

            it("when divisible by 5", function() {
              expect(f.convertToFizzBuzz(5)).to.be.equal("Buzz");
              expect(f.convertToFizzBuzz(10)).to.be.equal("Buzz");
            });

            it("when divisible by 15", function() {
              expect(f.convertToFizzBuzz(15)).to.be.equal("FizzBuzz");
              expect(f.convertToFizzBuzz(30)).to.be.equal("FizzBuzz");
            });

            it("when not divisible by 3, 5 or 15", function() {
                      expect(f.convertToFizzBuzz(4)).to.be.equal("4");
                      expect(f.convertToFizzBuzz(7)).to.be.equal("7");
            });
          });

  describe("convertRangeToFizzBuzz()", function() {
            it("returns in correct order", function() {
              expect(f.convertRangeToFizzBuzz("1", "3")).to.be.eql(["1", "2", "Fizz"]);
            });

            it("applies FizzBuzz to every number in the range", function() {
                var spy = sinon.spy(f, "convertToFizzBuzz");

                f.convertRangeToFizzBuzz("9", "50");

                for (var i = 9; i <= 50; i++) {
                  expect(spy.withArgs(i).calledOnce).to.be.eql(true, "Expected
convertToFizzBuzz to be called with " + i);
                }
                f.convertToFizzBuzz.restore();
              });
        });
});
```

test/cachefizzbuzz.test.js:

```
var sinon = require("sinon");
var CacheFizzBuzz = require("../cachefizzbuzz.js");

var testResult1 = {
    "from" : "1",
    "to" : "20",
    "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                 "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz" ]
};
```

```
var testResult2 = {
    "from" : "2",
    "to" : "21",
    "result" : ["2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz",
                "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz", "19", "Buzz",
"Fizz"]
};

var DBResult1 = {
    "total_rows" : 7,
    "offset" : 6,
    "rows" : [ {
      "id" : "35523244d141fb56c2e6b8dfa58d7fed",
      "key" : [ "1", "20" ],
      "value" : null,
      "doc" : {
        "_id" : "35523244d141fb56c2e6b8dfa58d7fed",
        "_rev" : "1-a0337a71e877726cb8fda7d6ceeb2bfc",
        "from" : "1",
        "to" : "20",
        "result" : [ "1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz",
                     "Buzz", "11", "Fizz", "13", "14", "FizzBuzz", "16", "17", "Fizz",
                     "19", "Buzz" ]
      }
    } ]
};

var DBResult2 = {
    "total_rows" : 7,
    "offset" : 7,
    "rows" : []
};

describe("CacheFizzbuzz", function() {
      var f = new CacheFizzBuzz("http://user:password@localhost/fizzbuzz");

      describe("dbStoreCalculatedResult()", function() {
        it("calls Cloudant to store the doc", function() {

          var mock = sinon.mock(f._Cloudant);
          mock.expects("insert").withArgs(testResult1).once();

          f._dbStoreCalculatedResult(testResult1);

          mock.verify();
          mock.restore();
        });
      });

      describe("fizzBuzzRange()", function() {
            var cbFunction = function(data) {
            };

            it("calls Cloudant to get record from the fb/range view in DB",
                function() {
                  var stub = sinon.stub(f._Cloudant, "view");

                  f.fizzBuzzRange("1", "20", cbFunction);

                  expect(stub.withArgs('fb', 'range', {
                    include_docs : true, key : [ "1", "20" ]
                  }, sinon.match.any).calledOnce).to.be
                  .eql(true,
                  "Expected cloudant.view to be called only once with correct
parameters");

                  f._Cloudant.view.restore();
                });
          });

      describe("processDBResult()", function() {
        var cbFunction = sinon.spy();
        var fizzBuzzSpy = null;

        beforeEach( function() {
          fizzBuzzSpy = sinon.spy(f._fizzbuzz, "convertRangeToFizzBuzz");
        });
```

```
        afterEach( function() {
          cbFunction.reset();
          f._fizzbuzz.convertRangeToFizzBuzz.restore();
        });

            it("processes the results from Cloudant with a valid result set",
                function() {
                  var storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult to parse DB return");
                  expect(fizzBuzzSpy).callCount(0);
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates the results when no results found then saves to DB",
                function() {
                  storeResultsStub = sinon.stub(f, "_dbStoreCalculatedResult");

                  f._processDBResult(false, DBResult2, "2", "21", cbFunction);

                  expect(cbFunction.withArgs(testResult2).calledOnce).to.be
                  .eql(true,"Expected calculated result when no data from DB");
                  expect(fizzBuzzSpy.withArgs("2", "21").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate
results");
                  expect(storeResultsStub.withArgs(testResult2).calledOnce).to.be
                  .eql(true, "Expect calculated restuls to be stored in DB");

                  f._dbStoreCalculatedResult.restore();
                });

            it("calculates results when DB error, but doesn't store results",
                function() {
                  storeResultsSpy = sinon.spy(f, "_dbStoreCalculatedResult");

                  f._processDBResult(true, DBResult1, "1", "20", cbFunction);

                  expect(cbFunction.withArgs(testResult1).calledOnce).to.be
                  .eql(true, "Expected processDBResult create empty result with DB
error");
                  expect(fizzBuzzSpy.withArgs("1", "20").calledOnce).to.be
                  .eql(true, "convertRangeToFizzBuzz should be called to calculate
results");
                  expect(storeResultsSpy).callCount(0);

                  f._dbStoreCalculatedResult.restore();
                });
          });

      //remaining tests go above here
});
```