
ORACLE®

From the Beginning: Your First Node.js Web Service

developer.oracle.com

P. Venkatramen
Data Access Development
Oracle Database
10 April 2018

Live for the Code

ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Me

P. Venkatraman

Principal Member, Technical Staff, Oracle

- Developer in Data Access team, Oracle DB
- node-oracledb development and other APIs

The Plan Today

1. What's What
2. Installation
3. The Web Service
4. Demonstration
5. Next Steps

1 What's What

2 Installation

3 The Web Service

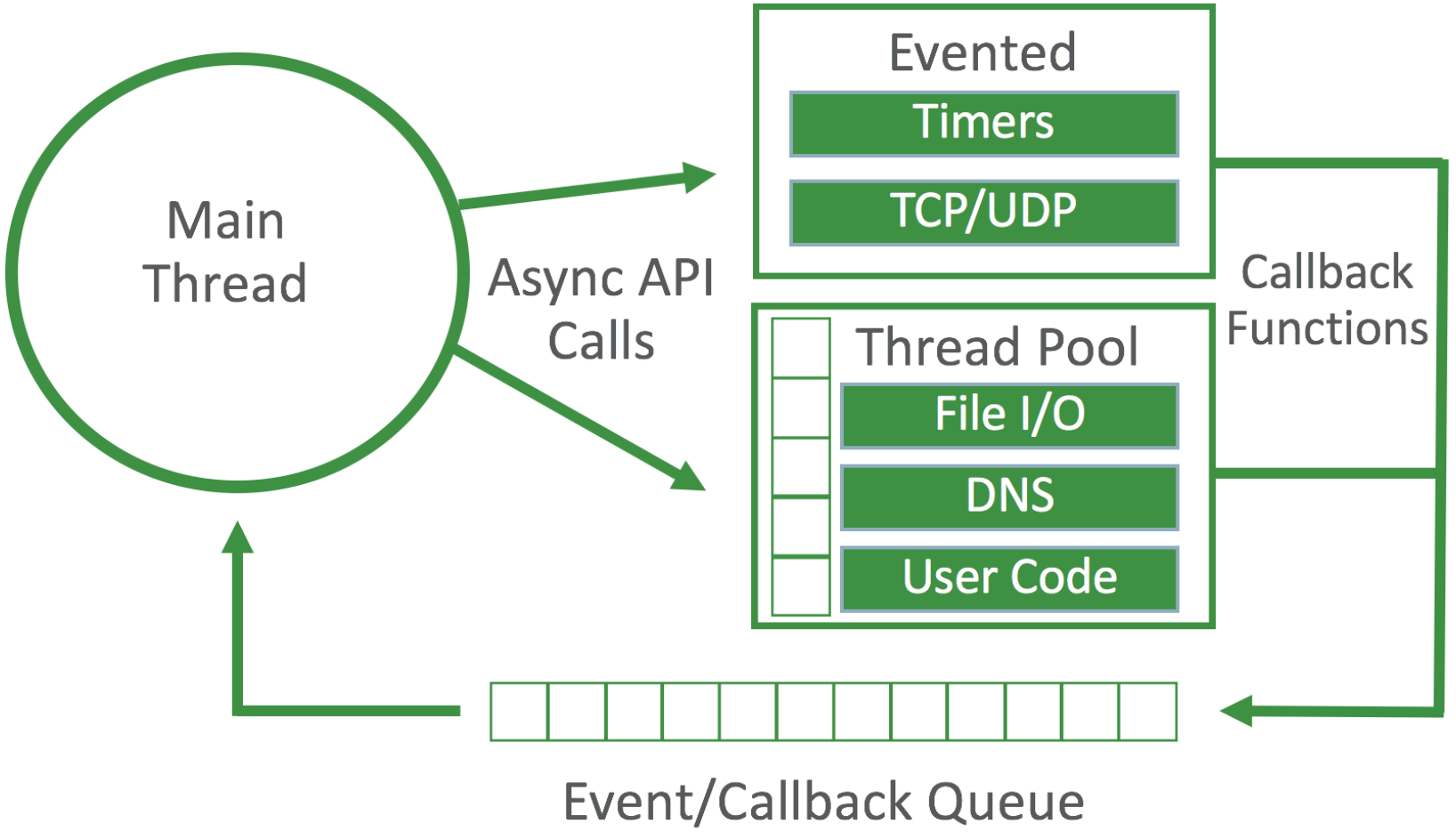
4 Demonstration

5 Next Steps

What is Node.js?

- Server-side JavaScript runtime
  - One language, front-end and back-end
- JavaScript runtime built on Chrome's V8 JavaScript engine
  - Though V8 is being decoupled to allow for other JavaScript engines
- Package ecosystem (NPM) is world's largest repo of open-source libraries
- Lightweight and efficient: event-driven, non-blocking I/O model
- Great for Web Services

Node.js architecture (not entirely accurate)



What is node-oracledb?

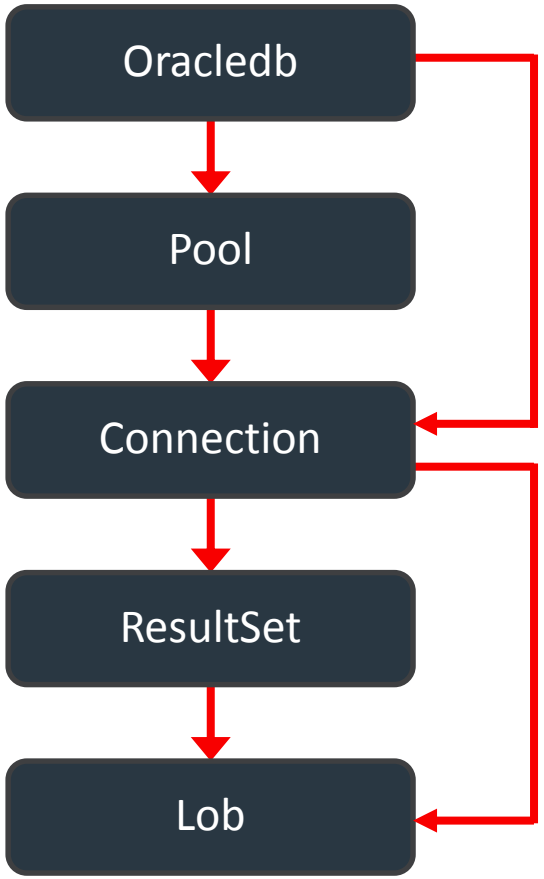
Base class
- Get connections or create pools
- Set configuration parameters

Connection Pooling
- Dramatically increases performance
- Built-in pool cache for convenience

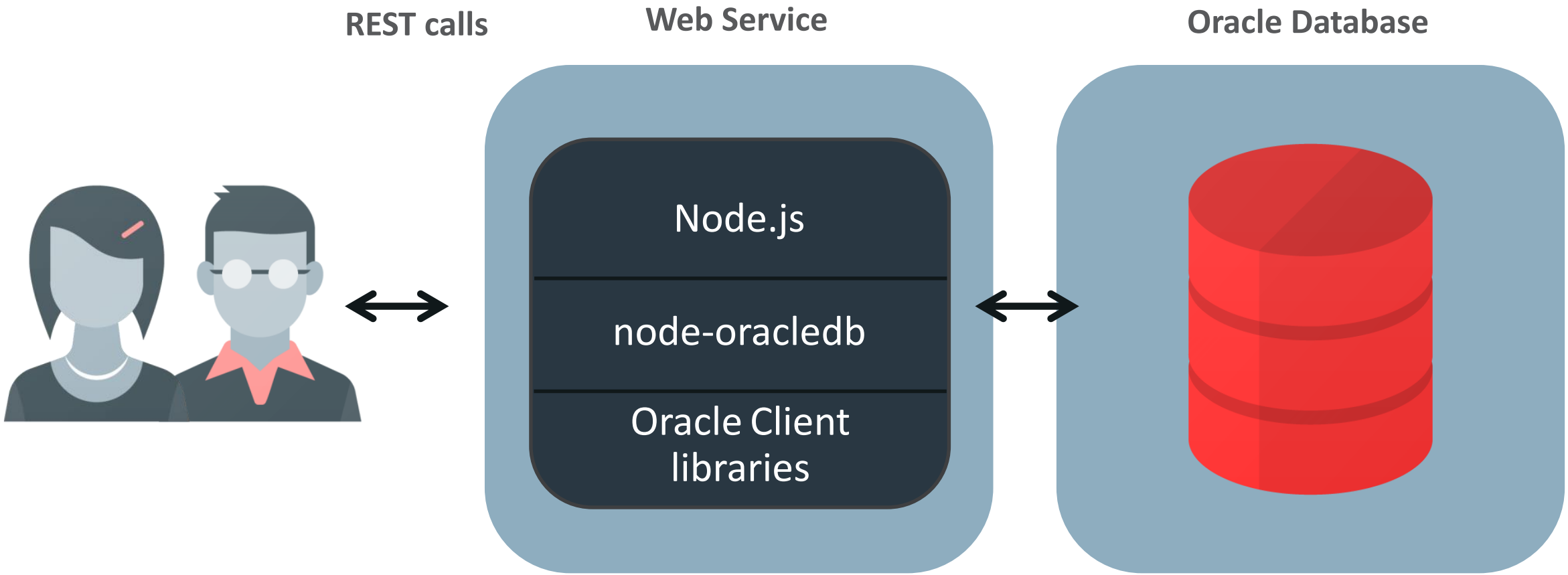
SQL and PL/SQL Execution
- Transaction support w/data type conversion
- Bind using JavaScript objects or arrays

Read-consistent, pageable cursor
- Used for large result sets
- Recursive callbacks or Node.js streams

Large object support
- Stream large LOBs with this class
- Can fetch smaller LOBs as string/buffer



The Goal Today

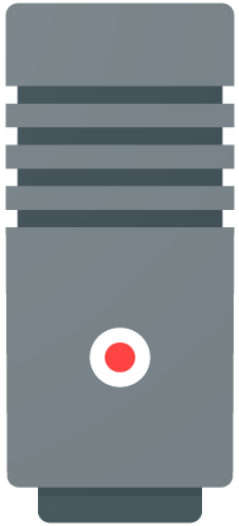


Basic ingredients for a REST API

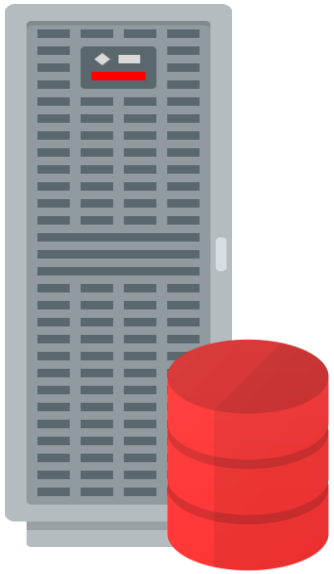
Client

Middle-tier

Web Server | Routing Logic | Controller Logic | Database Logic



Database



What features do you need to support?

- Pagination, sorting, & filtering
- Authentication & authorization
- Caching/ETag
- Doc/Swagger
- Real-time push/WebSockets
- Throttling
- Multiple representations (JSON, XML, CSV)
- CORS

1 What's What

2 Installation

3 The Web Service

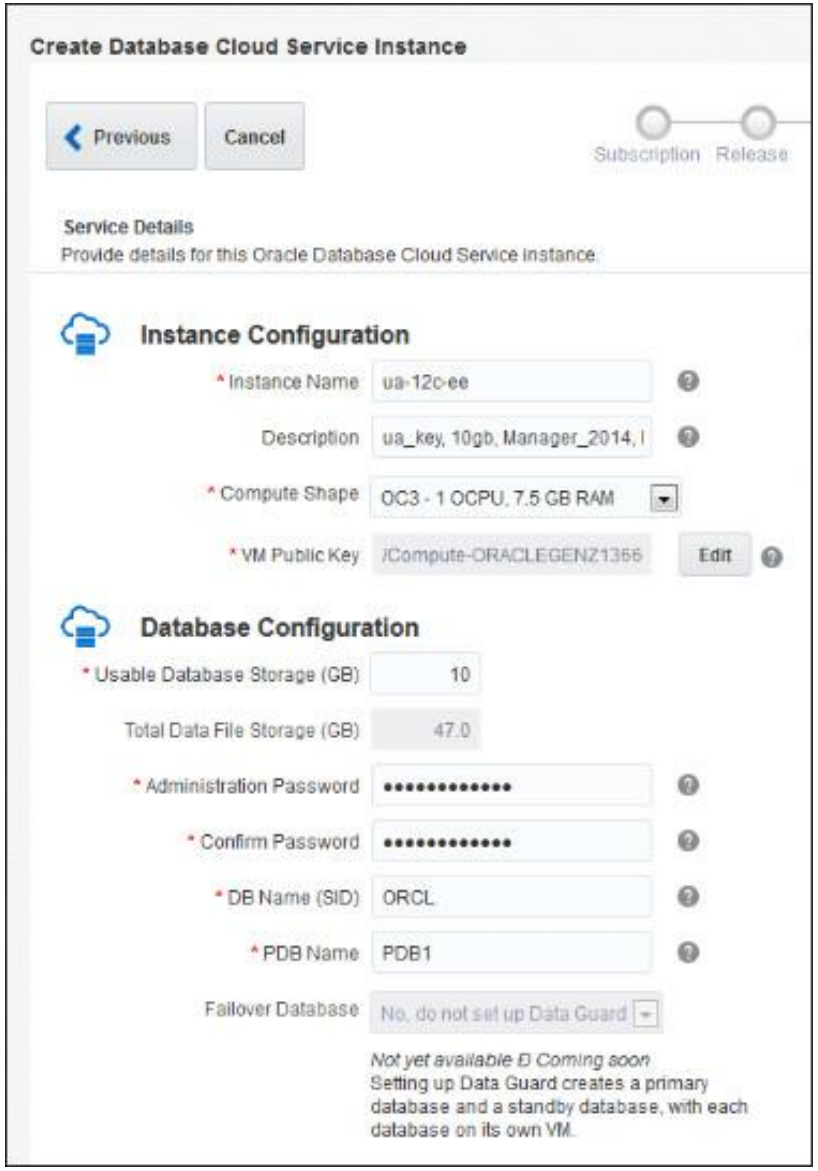
4 Demonstration

5 Next Steps

Installing Oracle DB

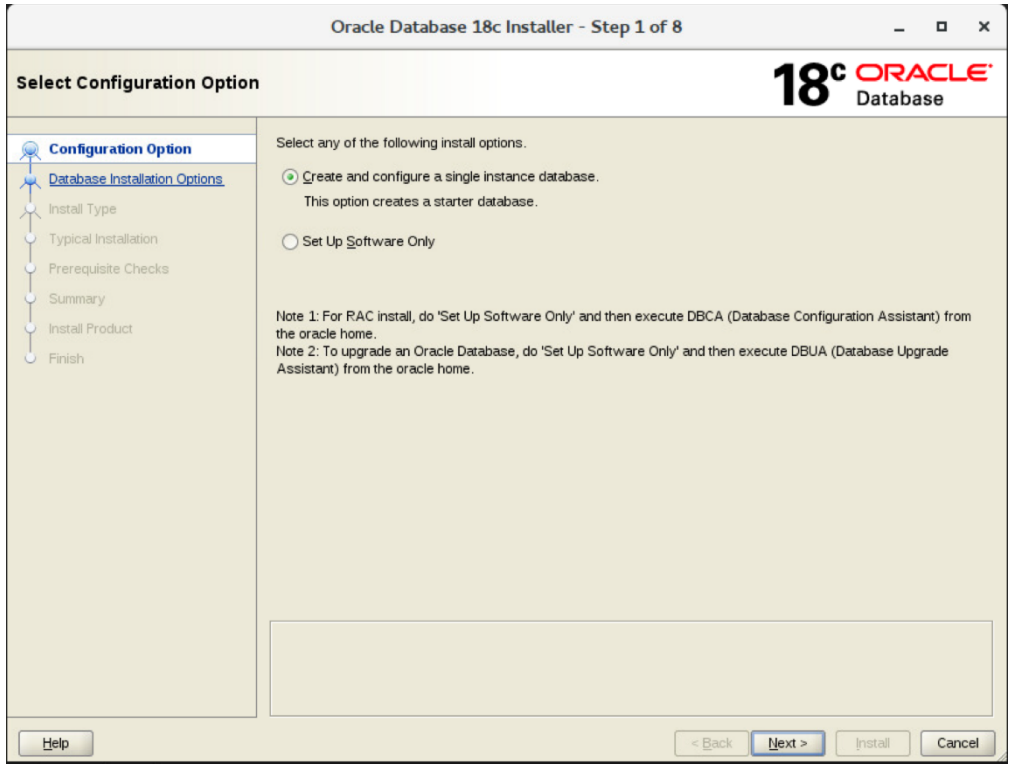
Use Oracle Cloud DBaaS

- Let someone do the install



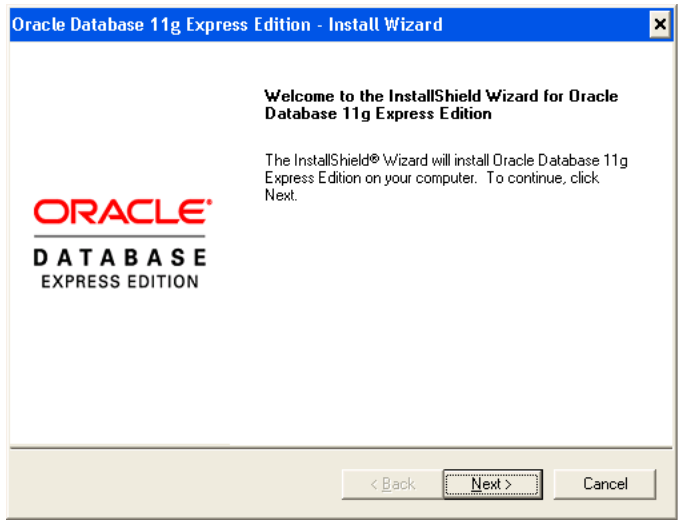
Install Oracle Enterprise Edition

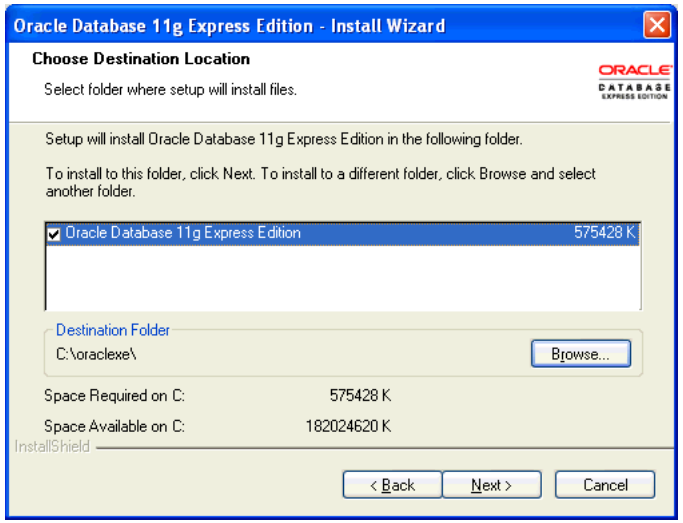
- Full Database Install
  - ./runInstaller

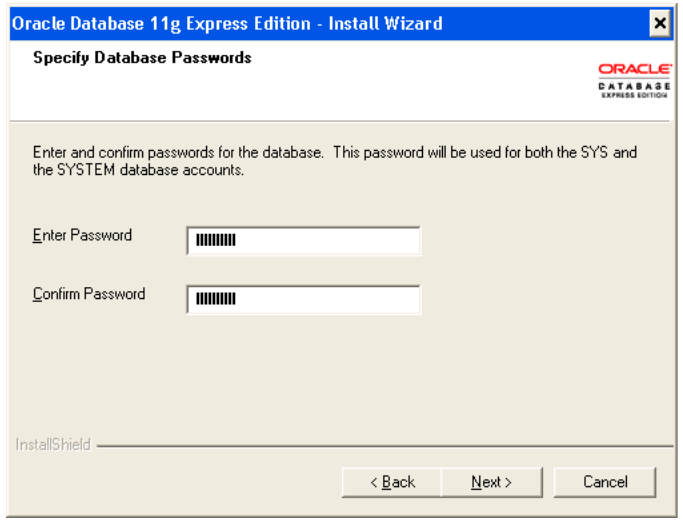


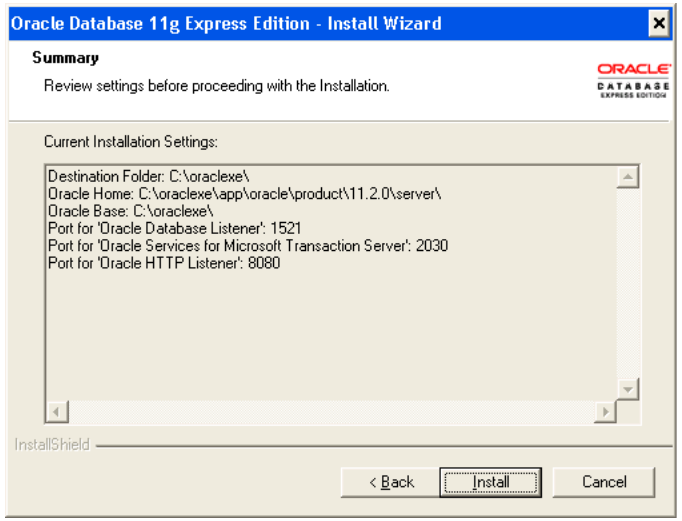
Install Oracle XE

- Linux & Windows
  - rpm or EXE
  - Simple install
- Latest is 11gR2



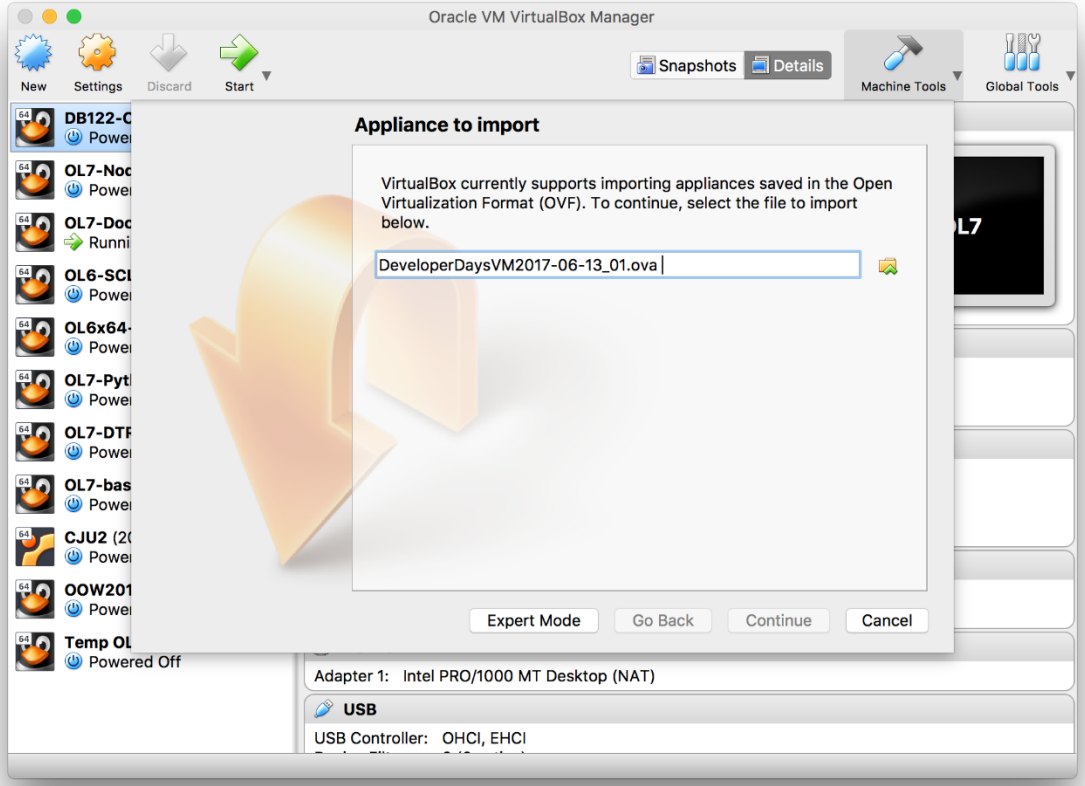


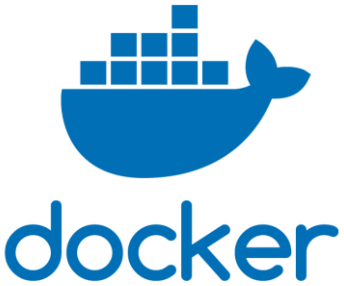




Use a Pre-built VirtualBox VM

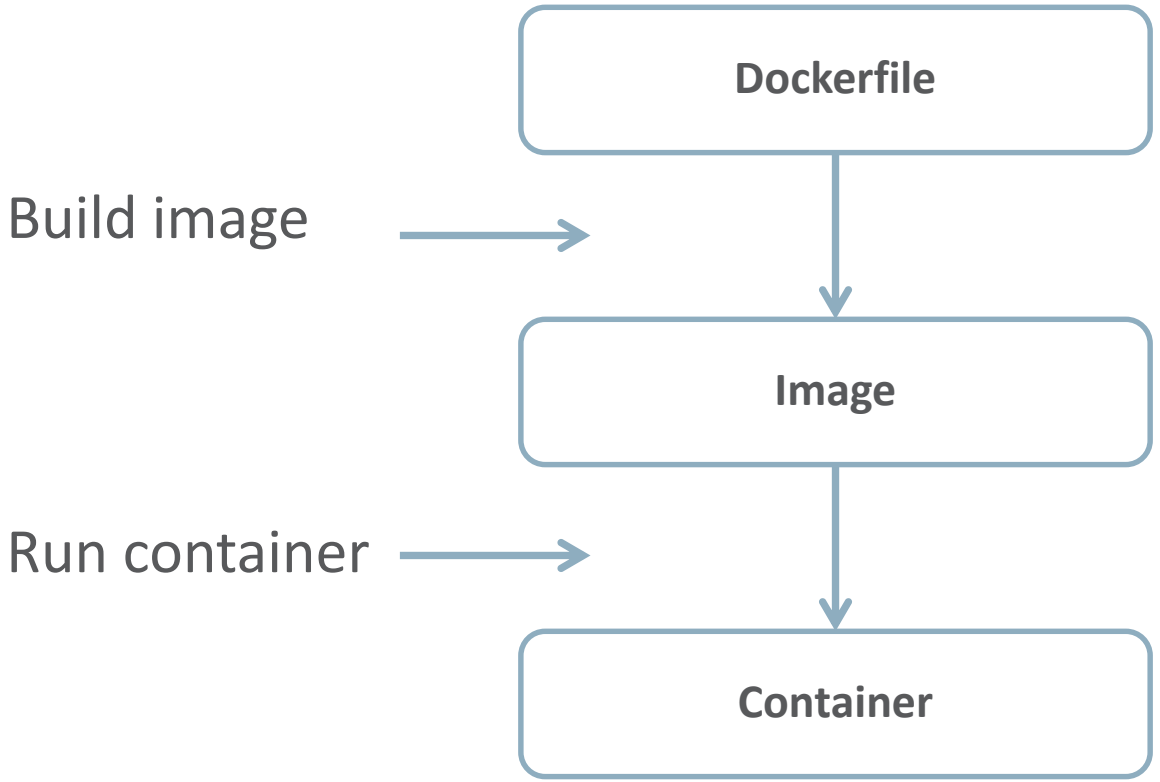
- App Development VM includes the DB and SQL Developer
  - http://www.oracle.com/technetwork/community/developer-vm/





But Let’s Use Docker!

Docker Images and Containers



Installing Docker

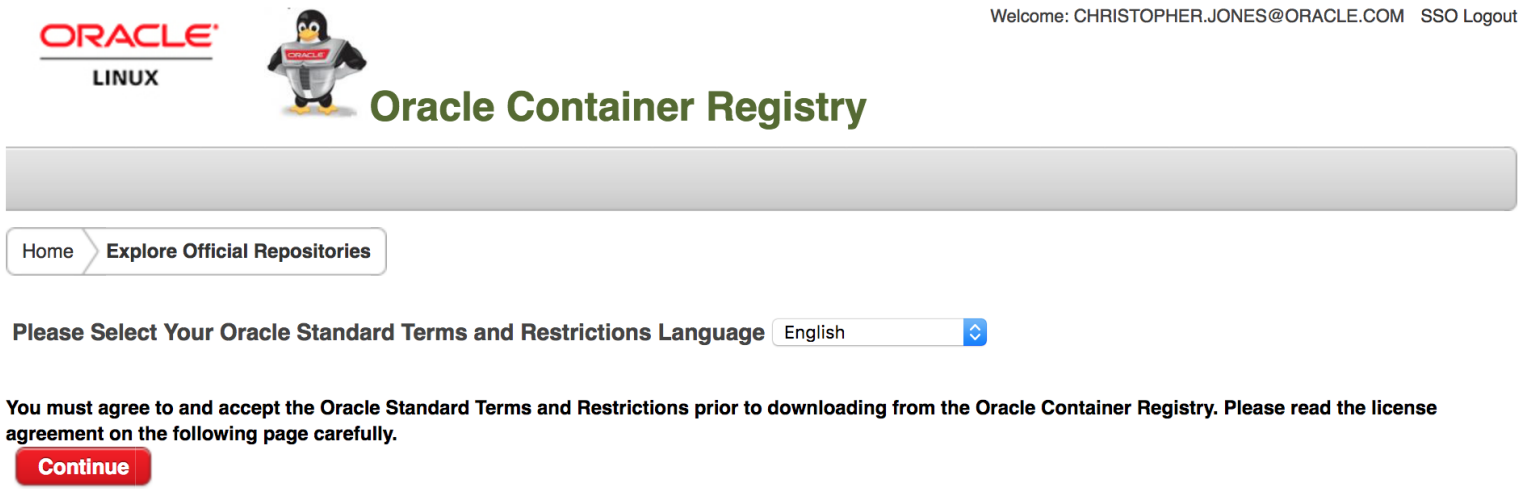
Install Docker on your Platform

- I'll use Oracle Linux 7
  - This has the ol7_latest and ol7_uekr4 channels already enabled
- Run (as the 'root' user)

```
# yum-config-manager --enable ol7_addons
# yum install docker-engine
# systemctl enable docker
# systemctl start docker
```

Oracle Docker Images

- Oracle images are available from
  - https://store.docker.com/
  - https://container-registry.oracle.com
- Accept the container-registry license



Get Oracle Docker Images

- Login from the host where Docker is installed

```
# docker login container-registry.oracle.com
```

- Oracle Database Enterprise Edition

```
# docker pull container-registry.oracle.com/database/enterprise:12.2.0.1
```

- Oracle Instant Client

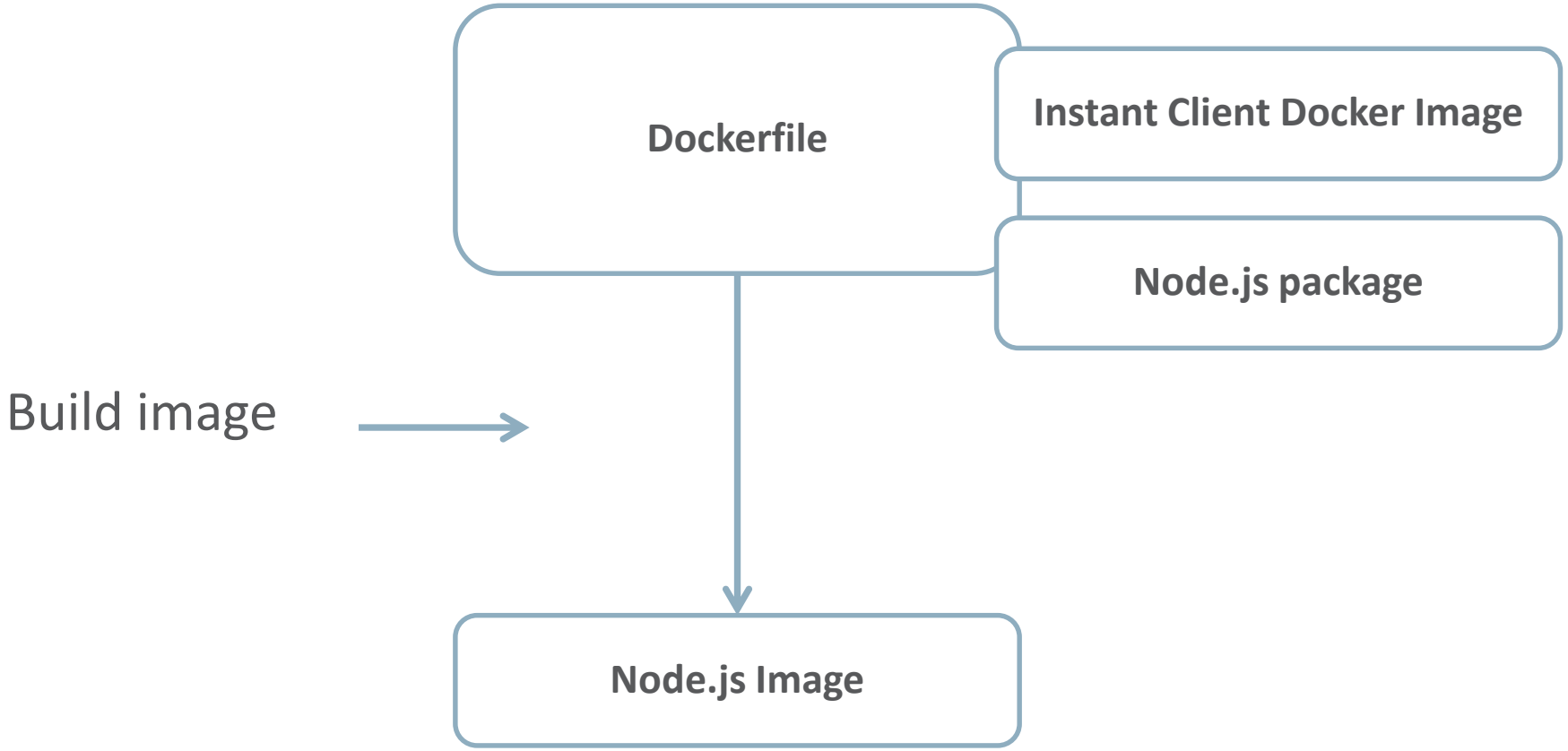
```
# docker pull container-registry.oracle.com/database/instantclient:12.2.0.1
```

- View the installed images

```
# docker images
REPOSITORY                                                TAG          IMAGE ID       CREATED          SIZE
container-registry.oracle.com/database/enterprise         12.2.0.1     12a359cd0528   7 months ago     3.44GB
container-registry.oracle.com/database/instantclient      12.2.0.1     fda46de41de3   7 months ago     407MB
```

Installing Node.js

Create a Node.js Docker Image



Create a Node.js Docker Image

- Oracle Linux RPM for several Node.js versions are in yum channels on

  http://yum.oracle.com/oracle-linux-nodejs.html

- Create ~cjones/nodejs-scripts/Dockerfile:

```
FROM container-registry.oracle.com/database/instantclient:12.2.0.1
ADD ol7_developer_nodejs8.repo /etc/yum.repos.d/ol7_developer_nodejs8.repo
RUN echo proxy=http://www-proxy.us.oracle.com:80 >> /etc/yum.conf
RUN yum -y install nodejs
```

- Create the new Node.js image based on the Instant Client image:

```
docker build -t cjones/nodejs-image ~cjones/nodejs-scripts/
```

Our Docker Images

```
# docker images
REPOSITORY                                                TAG          IMAGE ID       CREATED          SIZE
cjones/nodejs-image                                       latest       e048b739bb63   29 minutes ago   1.51GB
container-registry.oracle.com/database/enterprise         12.2.0.1     12a359cd0528   7 months ago     3.44GB
container-registry.oracle.com/database/instantclient      12.2.0.1     fda46de41de3   7 months ago     407MB
```

1 What's What

2 Installation

3 The Web Service

4 Demonstration

5 Next Steps

Banana Farmers

The Node.js Web Service

The Scenario

- Shipments of bananas from farmers are recorded
- Shipments have a farmer name, ripeness, and weight
- Shipments can be inserted, queried, updated or deleted

Banana Color Guide

1 

ALL GREEN

As received at your warehouse from Central and South America

2 

LIGH

3 

4 

5 

6 

FU

Firm fruit with great eating flavor. Easily bruised – handle with extra care.

7 

YELLOW FLECKED WITH BROWN

Sweet eating flavor. Perfect texture and consistency for blender drinks and baking.

The Schema

```
CREATE TABLE bananas (shipment VARCHAR2(4000) CHECK (shipment IS JSON));


INSERT INTO bananas VALUES (
       '{ "farmer": "Gita", "ripeness": "All Green", ”kilograms": 100 }');
INSERT INTO bananas VALUES (
       '{ "farmer": "Ravi", "ripeness": "Full Yellow", ”kilograms": 90 }');
INSERT INTO bananas VALUES (
       '{ "farmer": "Mindy", "ripeness": "More Yellow than Green", ”kilograms": 92 }');
```

Manipulating the JSON Schema

- **`SELECT b.shipment FROM bananas b WHERE b.shipment.farmer = 'Gita'`**
  - Gives **`{"farmer" : "Gita", "ripeness" : "All Green", "kilograms" : 100 }`**
  - This is a REST GET
- **`INSERT INTO bananas (shipment) VALUES (:s)`**
  - This is a REST POST
- **`UPDATE bananas b SET shipment = :s WHERE b.shipment.farmer = :f`**
  - This is a REST PUT
- **`DELETE FROM bananas b WHERE b.shipment.farmer = :f`**
  - This is a REST DELETE

The Web Service

File ‘package.json’

```
. . .
"dependencies": {
   "body-parser": "^1.18.2",
   "express": "^4.16.0",
   "oracledb": "^2.2.0"
 },
 "main": "server.js
. . .
```

The Web Service

File ‘server.js’

```
var express = require('express');
var bodyParser = require('body-parser');
var oracledb = require('oracledb');
var dbConfig = require('./dbconfig.js');

var app = express();
app.use(bodyParser.json());

. . .
```

Connection

```
oracledb.createPool({
      user: dbConfig.user,
      password: dbConfig.password,
      connectString: dbConfig.connectString
    }, . . .
```

Connection Helper

```
function doGetConnection(res, cb) {
  oracledb.getConnection(function (err, connection) {
    if (err) {
      res.set('Content-Type', 'application/json');
      res.status(500).send(JSON.stringify({
        status: 500,
        message: "Error getting DB connection",
        detailed_message: err.message
      }));
    } else
      cb(err, connection);
  });  }
```

The GET handler (1)

/bananas/:FARMER

```
app.get('/bananas/:FARMER', function (req, res) {
  doGetConnection(res, function(err, connection) {
    if (err)
      return;
    connection.execute(
      "SELECT b.shipment FROM bananas b WHERE b.shipment.farmer = :f",
      { f: req.params.FARMER },
      function (err, result) {
       . . .
```

The GET handler (2)

/bananas/:FARMER

```
if (err) {
  res.set('Content-Type', 'application/json');
  res.status(500).send(JSON.stringify({
    status: 500,  message: "Error getting the farmer's profile",
    detailed_message: err.message
  }));
} else if (result.rows.length < 1) {
  res.set('Content-Type', 'application/json');
  res.status(404).send(JSON.stringify({
    status: 404, message: "Farmer doesn't exist", detailed_message: ""
  })); }
```

The GET handler (3)

/bananas/:FARMER

```
        else {
          res.contentType('application/json');
          res.status(200).send(JSON.stringify(result.rows));
        }
        doRelease(connection, "GET /bananas/" + req.params.FARMER);
        });
    });
});
```

All Web Service Routes

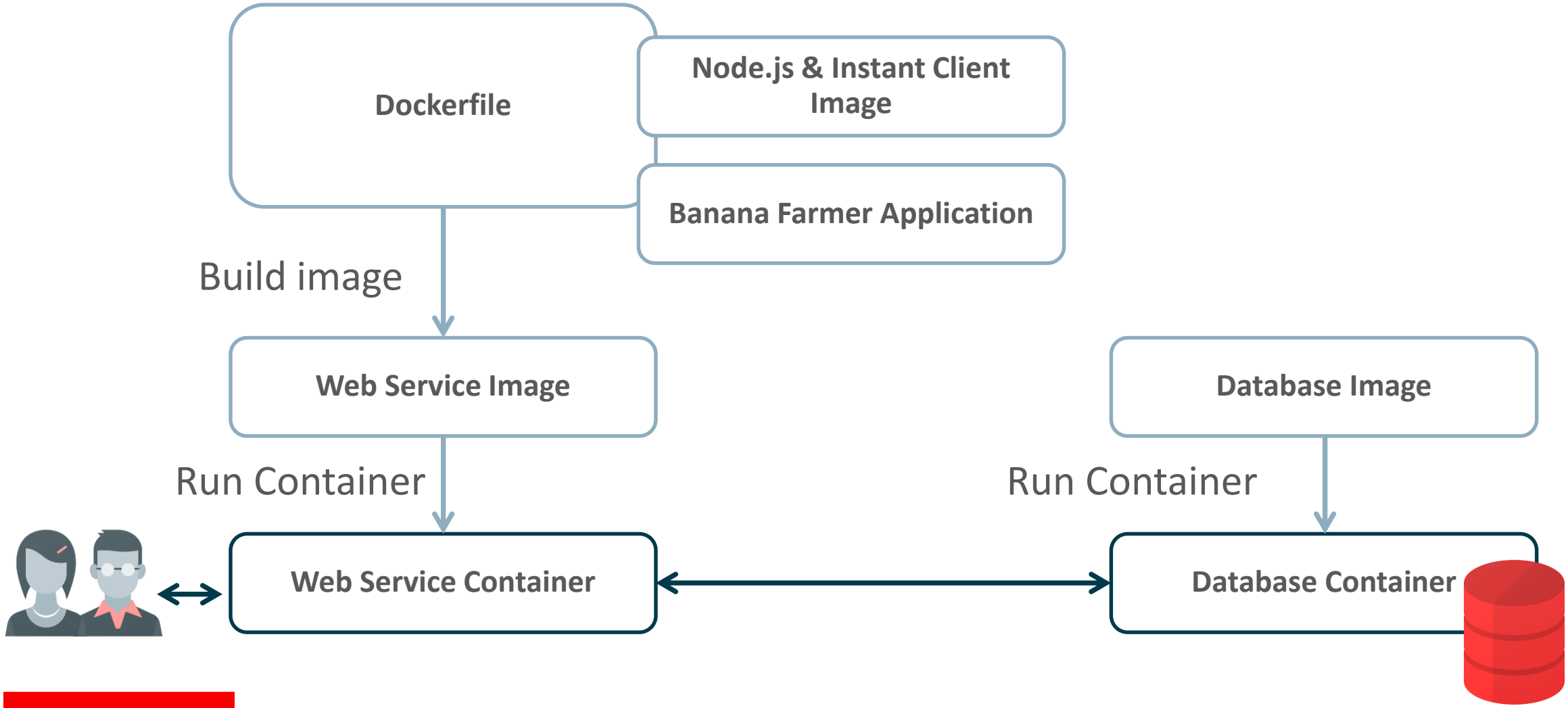
```
// GET: Get all banana shipments
app.get('/bananas', function (req, res) { . . . }

// GET: Get the banana shipment for a farmer
app.get('/bananas/:FARMER', function (req, res) { . . . }

// POST: Create a new banana shipment for a farmer
app.post('/bananas', function (req, res) { . . . }

// PUT: Update the banana shipment for a farmer
app.put('/bananas/:FARMER', function (req, res) { . . . }

// DELETE: Delete banana shipments for a farmer
app.delete('/bananas/:FARMER', function (req, res) { . . . }
```

Standalone Testing

- **`npm install`**
- Edit **`dbconfig.js`** and set credentials and connection string
- **`npm start`** (which runs '**`node server.js`**')
- Use the Web Service

Putting it Together

Putting it Together



Web Service File Overview

`ws-demo-scripts` directory:

`Dockerfile`	The recipe to build a Docker image
`package.json`	Node.js application configuration
`server.js`	Web Service Code
`dbconfig.js`	Connection credentials
`run.sh`	The command to start the WS
`envfile.list`	Credential environment variables
`createschema.sql`	Bootstrap data values

Build the Web Service Image

```
# docker build -t cjones/ws-demo ~cjones/ws-demo-scripts
```

~cjones/ws-demo-scripts/Dockerfile contains:

```
FROM cjones/nodejs-image
ENV http_proxy=http://www-proxy.us.oracle.com:80 \
    https_proxy=http://www-proxy.us.oracle.com:80
RUN mkdir workdir
WORKDIR workdir
COPY package.json package.json
COPY server.js server.js
COPY dbconfig.js dbconfig.js
COPY run.sh run.sh
RUN npm install
CMD ["./run.sh"]
```

Our Docker Images

```
# docker images
REPOSITORY                                                TAG          IMAGE ID       CREATED              SIZE
cjones/ws-demo                                            latest       31cbe6d2ea4e   21 seconds ago       1.51GB
cjones/nodejs-image                                       latest       e048b739bb63   29 minutes ago       1.51GB
container-registry.oracle.com/database/enterprise         12.2.0.1     12a359cd0528   7 months ago         3.44GB
container-registry.oracle.com/database/instantclient      12.2.0.1     fda46de41de3   7 months ago         407MB
```

The Database Container

- Start the Database Container:

```
# docker run –d --name demodb -P \
        container-registry.oracle.com/database/enterprise:12.2.0.1
```

- Check status until it shows ‘(healthy) ’:

```
# docker ps
```

- Find the IP address so we can connect the Web Service to the DB:

```
# docker inspect -f "{{ .NetworkSettings.IPAddress }}" demodb
```

- The Container can be stopped/started as desired with:

```
# docker stop demodb
# docker start demodb
```

Create the Banana Farmer Schema

```
# sqlplus -l sys/Oradoc_db1@172.17.0.2/orclpdb1.localdomain as sysdba \
    @ createschema.sql

SQL> CREATE USER scott IDENTIFIED BY tiger;
SQL> GRANT CONNECT, RESOURCE TO scott;
SQL> ALTER USER scott QUOTA UNLIMITED ON USERS;
SQL> CREATE TABLE scott.bananas (
  2  shipment VARCHAR2(4000) CHECK (shipment IS JSON));
SQL> INSERT INTO scott.bananas VALUES (
  2  '{"farmer": "Gita", "ripeness": "All Green", "kilograms": 100}');
. . .
```

The Web Service Container

- Set DB IP in the connect string environment variable in
  ~cjones/ws-demo-scripts/envfile.list:

```
NODE_ORACLEDB_CONNECTIONSTRING=172.17.0.2/orclpdb1.localdomain
```

  – This is used in dbconfig.js when the container runs
- Start the Web Service Container:

```
# docker run -d --name nodejs -P \
    --env-file ~cjones/ws-demo-scripts/envfile.list cjones/ws-demo
```

- Find the Web Service IP address:

```
# docker inspect -f "{{ .NetworkSettings.IPAddress }}" nodejs
```

- The Container can be stopped/started as desired:

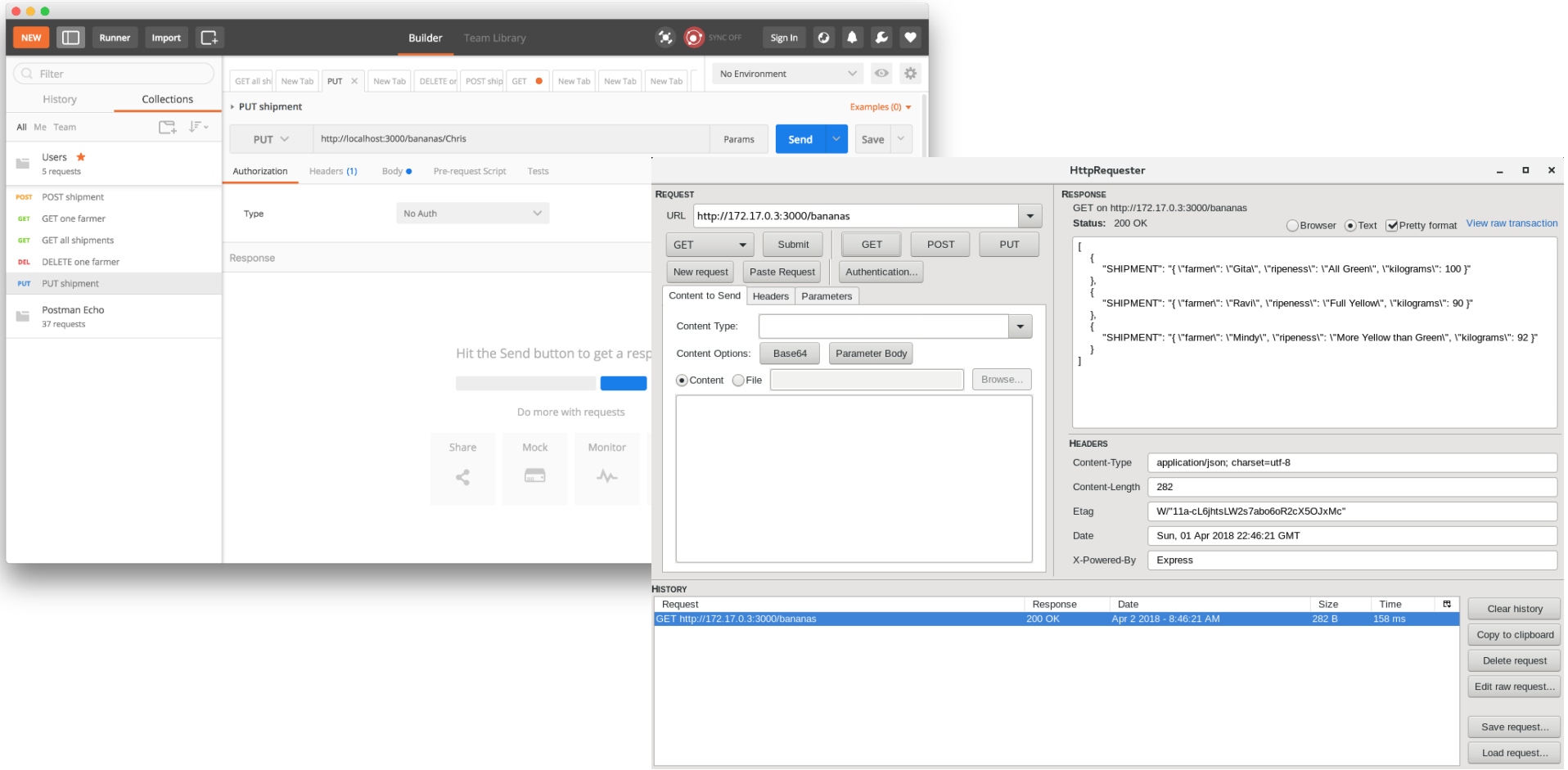
  docker stop nodejs or **docker start nodejs**

Done!

```
# docker images
REPOSITORY                                          TAG          IMAGE ID       CREATED             SIZE
cjones/ws-demo                                      latest       25caede29b17   12 minutes ago      1.51GB
cjones/nodejs-image                                 latest       138f2b76ffe7   13 minutes ago      1.51GB
container-registry.oracle.com/database/enterprise   12.2.0.1     12a359cd0528   7 months ago        3.44GB
container-registry.oracle.com/database/instantclient 12.2.0.1    fda46de41de3   7 months ago        407MB

# docker ps
CONTAINER ID  IMAGE                COMMAND                  STATUS                PORTS                                                 NAMES
2924e1225290  cjones/ws-demo       "./run.sh"               Up 3 hours                                                                  nodejs
9596bc2345d3  [...]/database/[...] "/bin/sh -c '/bin/..." Up 3 hours (healthy) 0.0.0.0:32803->1521/tcp, 0.0.0.0:32802->5500/tcp demodb
```

- The Web Service (in 'nodejs') knows the connection string for the database (in 'demodb')
  - **`172.17.0.2/orclpdb1.localdomain`**
- We know the IP of the web service from the 'docker inspect' command
  - **`172.17.0.3`**
- We know the Web Service port and endpoints (coded in **`server.js`**)
  - **`http://172.17.0.3:3000/bananas`**

Install Postman or HttpRequester Browser Extension



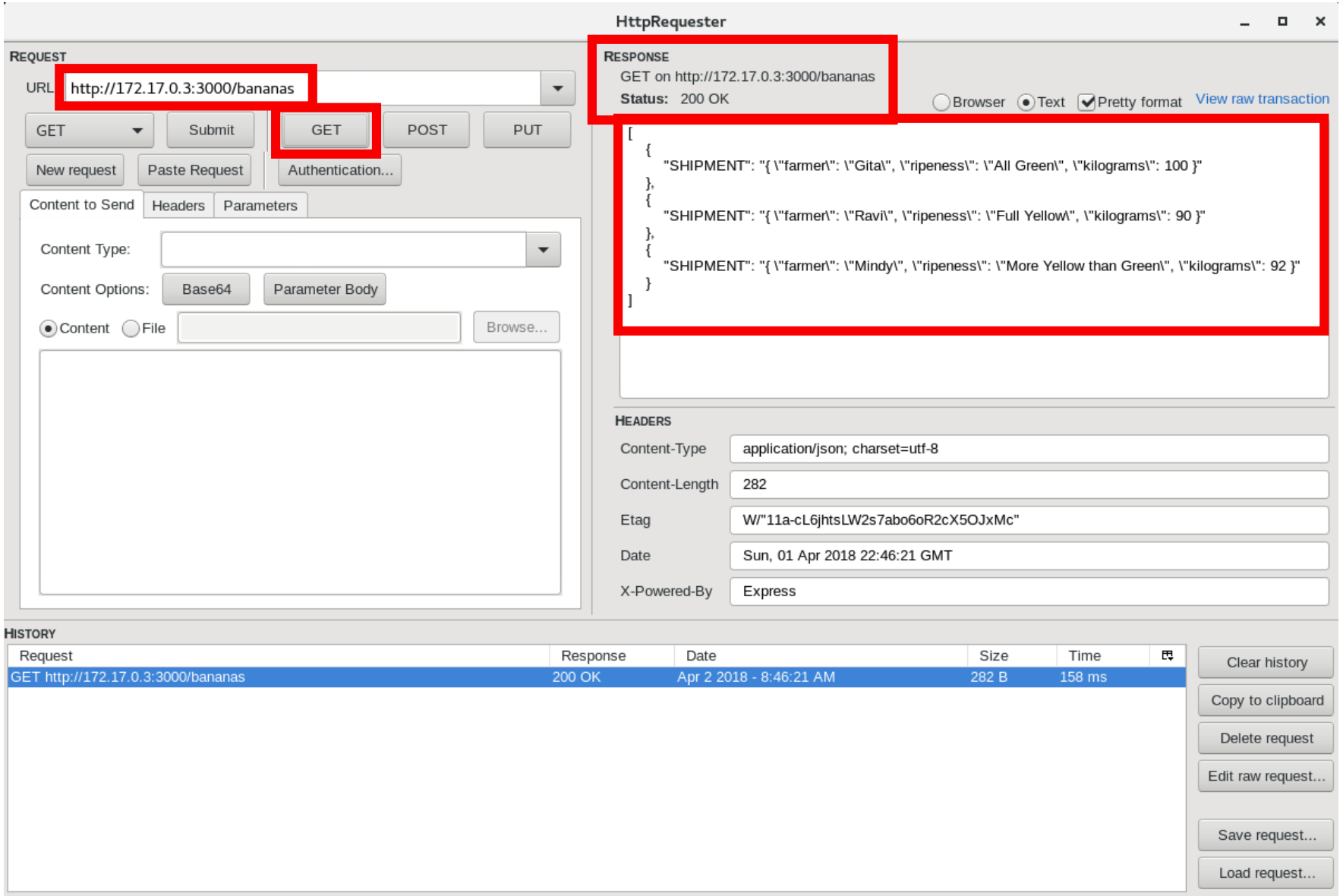
1 What's What

2 Installation

3 The Web Service

4 Demonstration

5 Next Steps



1 What's What

2 Installation

3 The Web Service

4 Demonstration

5 Next Steps

Next Steps

- Explore node-oracledb features
  - https://oracle.github.io/node-oracledb/
- Customize your own Database image from Oracle Docker scripts
  - https://github.com/oracle/docker-images
- Read the Blog series “Creating a REST API with Node.js and Oracle Database”
  - https://jsao.io/2018/03/creating-a-rest-api-with-node-js-and-oracle-database/
- Explore Sails, Loopback, GraphQL, . . .

References

Homepage	**https://oracle.github.io/node-oracledb/**
Help	**https://github.com/oracle/node-oracledb/issues**
Christopher Jones	**@ghrd**
	https://blogs.oracle.com/opal
Dan McGhan	**@dmcghan**
	https://jsao.io/
Anthony Tuininga	**@AnthonyTuininga**
P. Venkatraman	**@pvenkatraman**

ORACLE

Integrated Cloud

Applications & Platform Services


ORACLE®