**Creating Azure Web App**

1. Log in to Azure Portal https://portal.azure.com/,

2. Create Resource Group for Node JS website related resources (more on resource groups: https://docs.microsoft.com/pl-pl/azure/azure-resource-manager/resource-group-overview),

3. Create new Azure IoT Hub and assign it to the resource group created in the previous step,

4. Provide following settings in the Web App configuration:

   - Name – unique domain name for the website,
   - Publish – Code,
   - Runtime stack – most recent version of NodeJS,
   - Operating System – Linux,
   - Region – select region used for Resource group,

5. Provide App Service Plan name and select B1 in pricing tier options (read more about service plans at: https://docs.microsoft.com/pl-pl/azure/app-service/overview-hosting-plans),

6. Wait until Azure creates your web app and make sure it's available in the CLI using following command:

   *az webapp list --query [].name*

7. Try to reach your web application using following website address to see if it's already available (replace %websitename% with your web app name):
   http:// %websitename%.azurewebsites.net/

**Creating Node.js express application**

1. Install Node.js (https://nodejs.org/en/)

2. Install Express generator package https://expressjs.com/en/starter/installing.html

   *npm install express-generator -g*

3. Create directory for the application and run "express" command in this directory to create default express application skeleton,

4. Run "npm install" in the directory with the application to restore node packages,

5. Run "npm start" to launch application, try to access it from web browser using "http://localhost:3000/" URL,

6. Type “code .” in the command line (when Node.js app folder is active) to launch Visual Studio Code and make application folder active,

7. Go to .\routes\index.js file, and change following line:

```
res.render('index', { title: 'Express' });
```

to:

```
res.render('index', { title: 'Azure' });
```

8. Run application again and check if above update got applied,

**Deploy Node.js express application to Azure and configure build server**

1. Create “.gitignore” file in the application folder,

2. Populate .gitignore file with the content that can be found at https://github.com/github/gitignore/blob/master/Node.gitignore

3. Run “git init” command in the Node.js application folder,

4. Go to Deployment Center tab in the WebApp settings (that got created in the first step), https://channel9.msdn.com/Shows/Azure-Friday/An-overview-of-Azure-App-Service-Deployment-Center

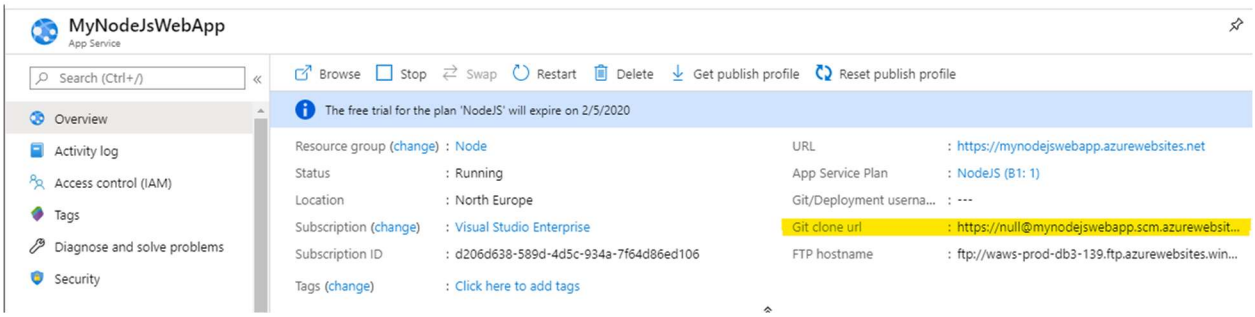
5. Select “Local Git” option and select “Kudu” build server,

6. Add remote repository by providing following command in the command line:

   *git remote add %NameOfRemoteRepo% %RepoURL%*

   *e.g. git remote add azureprod https://nodejs.scm.azurewebsites.net:443/mynodejswebapp.git*

   Please note: Git clone URL can be found in the overview section for our application in the Azure portal:




7. Push local content to remote repositor, following commands have to be executed:
   - Git add .
   - Git commit -m “Your commit message”
   - Git push *%remote repo name%* master

Please note: When pushing to remote repo Git will ask for credentials, these can be found in the “Deployment Center” for our application (in the Azure portal).

8. Try to reach your web application using following website address to see the new website is available (replace %websitename% with your web app name):
   http:// %websitename%.azurewebsites.net/

**Application slots**

https://docs.microsoft.com/en-us/azure/azure-functions/functions-deployment-slots

1. Update routes/index.js file in website application to pass more information to main view (views/index.jade):

```
var model = {
    title: 'Azure',
    message: process.env.MESSAGE || "This is development"
  }
res.render('index', model);
```

2. Update views/index.jade to use newly added the “message” property. Add following line at the end:

```
p #{message}
```

3. (Azure portal) Create two additional application settings in the Web App configuration settings:
   - MESSAGE = “This is production” (select option “Deployment slot setting”)
   - NODE_ENV = “production”
4. (Azure CLI) Crate new deployment slot by using following command (it’s necessary to provide additional parameters):

*az webapp deployment slot create*

Real example: az webapp deployment slot create -g Node -n MyNodeJsWebApp -s staging --configuration-source MyNodeJsWebApp

5. Copy git clone URL for newly created slot and add it as a remote for web app local repository. *git remote add %name of remote% %remote url%*

*example: remote add stage https://mynodejswebappstaging.scm.azurewebsites.net:443/mynodejswebapp.git*

6. Commit local changes that have been done locally (*git commit -a -m "**commit message**".*

7. Push change to "stage" repository: git push *%name of remote% master.*

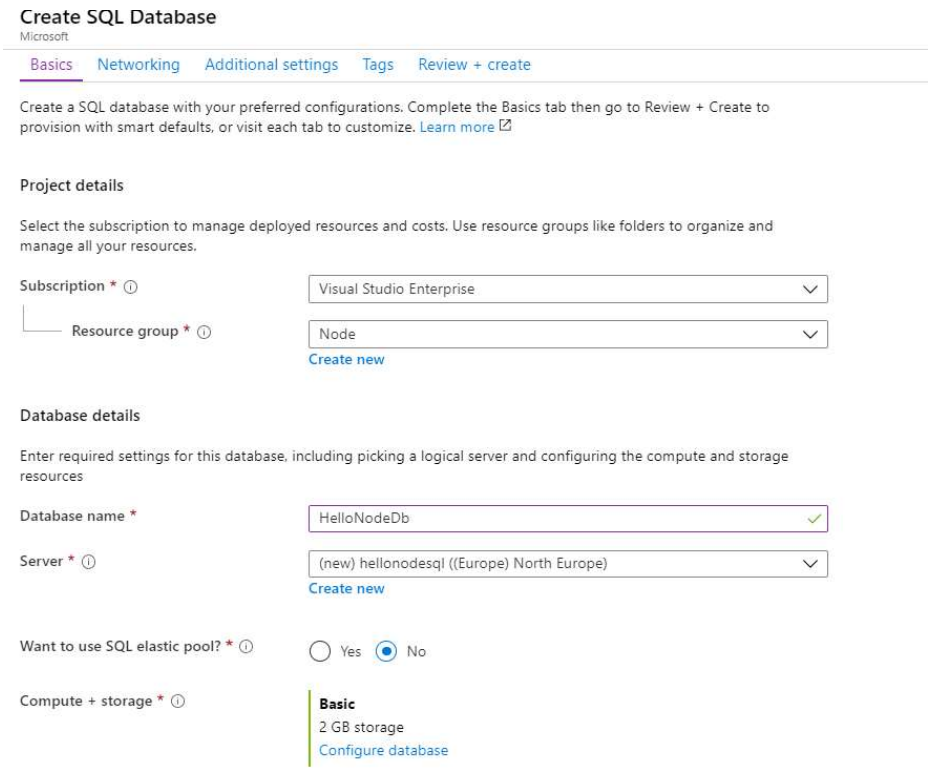
8. Open staging version of web application and compare it with production version.

9. Try to change "This is development" to "This is staging" in the stage slot (without changing code).

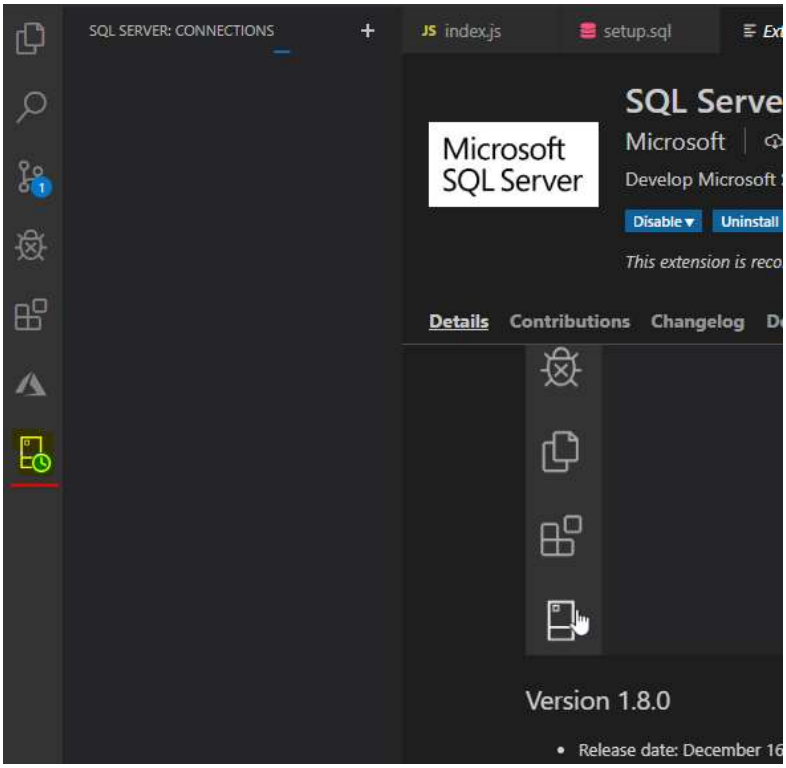
**Using Cloud Databases**

Azure SQL Database (https://azure.microsoft.com/pl-pl/services/sql-database/)

1. Create new Azure SQL database in the Azure portal, assign it to the resource group created for Node.js application, while creating Azure SQL database it's also required to create Server (create new server, make sure it's located in the same data center as other resources in the Node.js application). Select "No" in the SQL elastic pool question and Basic in the Compute + Storage option.

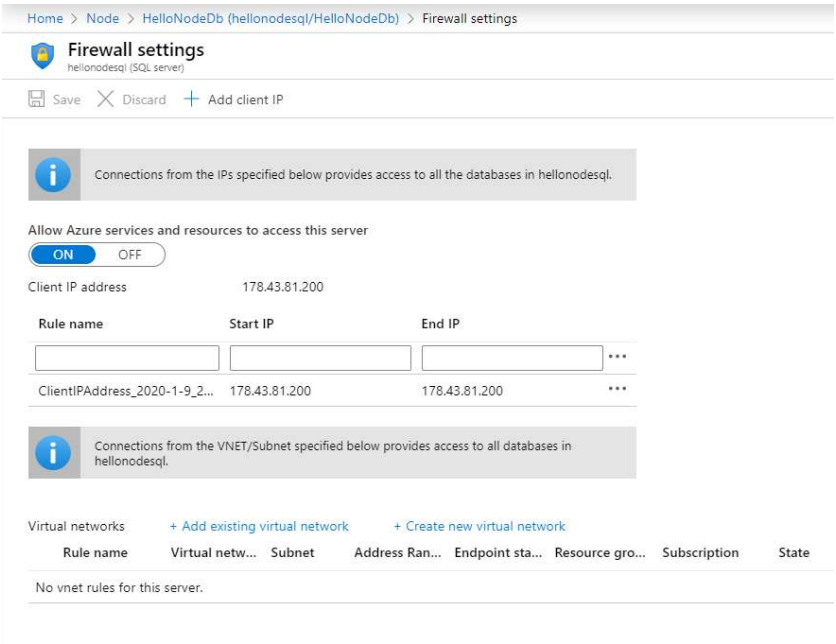



2. Add mssql extension to Visual Studio Code, when it's installed VS Code will get new tab added:

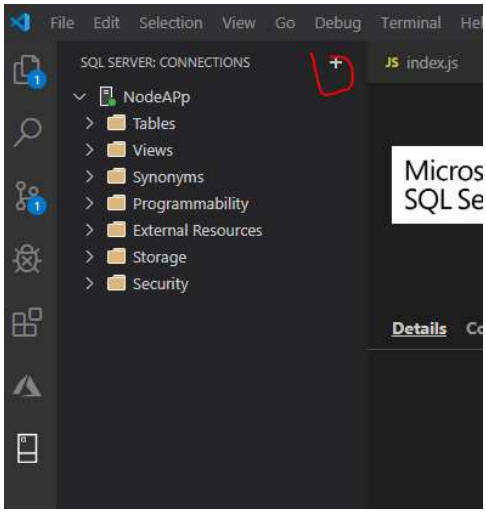


3. Create setup.sql file in the Node.js application folder,

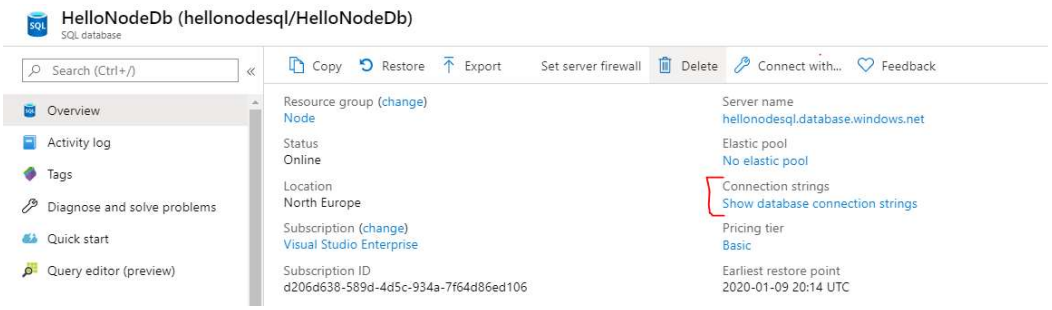
4. Enable App services connection + Add Client IP in the Firewall settings for the created database:




5. Connect to the database using mssql Visual Studio code extension.
    a. Add Connection




    b. Copy ADO.NET connection string from Azure portal (change password to the one set while creating Azure SQL server):




6. Create SQL user that will be used to connect to the database from Node.js application. Add following lines to setup.sql file created earlier:

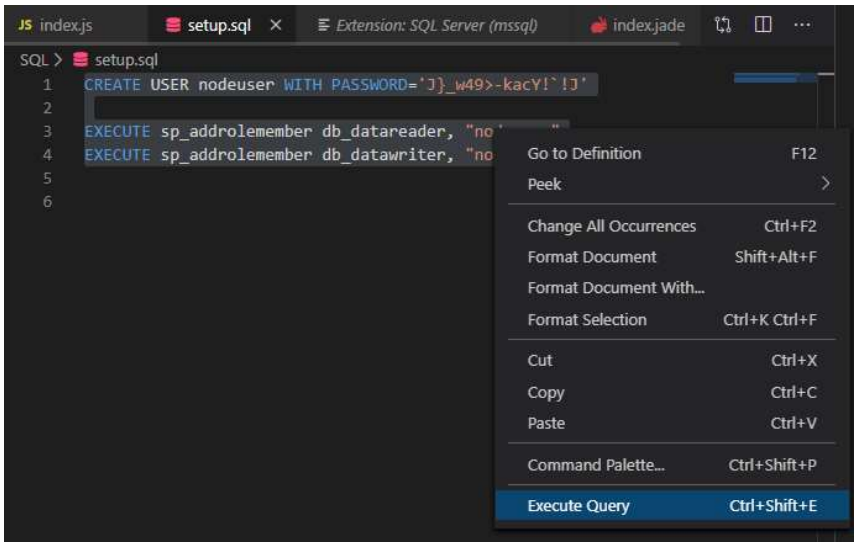
```
CREATE USER %user_name% WITH PASSWORD='%user_password%'

EXECUTE sp_addrolemember db_datareader, "%user_name% "
EXECUTE sp_addrolemember db_datawriter, "%user_name% "
```

Remember to replace:

%user_name% - with the user name that will be used later in Node.js application,

%user_password% - user password, in order to meet password policy it can be easier to use some network password generator e.g. https://passwordsgenerator.net/

7. Select above lines and execute them using context menu “Execute query” option, use connection created earlier in the step 5,




8. Create users table in the database using following code:

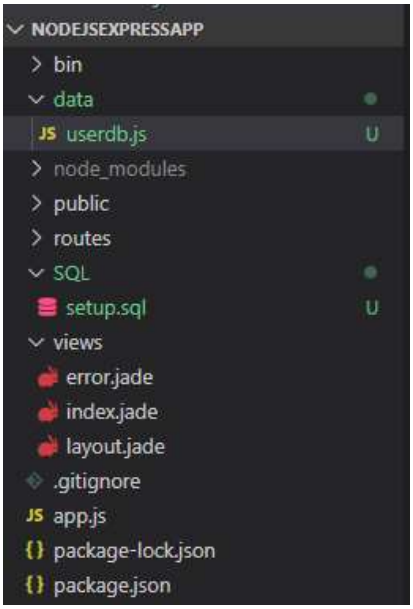
```
CREATE TABLE users
(
    id INT IDENTITY PRIMARY KEY,
    name nvarchar(255),
    email nvarchar(255)
)
```

9. Execute above query by selecting it and using “Execute query” context menu option,

10. Make sure that table is created by executing query “select * from users”.

11. Add new folder called “data” to Node.js application,

12. Add new file called “userdb.js” to the folder created in the previous step,




13. Add “tedious” module to Node.js application by executing “npm install tedious –save” from the application folder (read more about tedious at https://github.com/tediousjs/tedious),

14. Add following code to the userdb.js file:

```
var tedious = require('tedious');
var Connection = tedious.Connection;
var Request = tedious.Request;

var config = {
    server: 'hellonodesql.database.windows.net',
    options: {
        database: 'HelloNodeDb',
```

```
        encrypt: true,
        rowCollectionOnRequestCompletion: true
    },
    authentication: {
        options: {
            userName: 'nodeuser',
            password: 'J}_w49>-kacY!`!J'
        },
        type: 'default'
    }
}

var createUsers = function(callback) {
    var connection = new Connection(config);
    connection.on('connect', function(err){
        if(err){
            callback(err);
        }
        else {
            var request = new Request(
                `
                    INSERT INTO users(name, email) VALUES ('Jan', 'jan.kowalski@email.com')
                    INSERT INTO users(name, email) VALUES ('Tomasz', 'tomasz.kowalski@email.com')
                `,
                function(err, rowCount){
                    callback(err, rowCount);
                }
            );
            connection.execSql(request);
        }
    })
};

var queryUsers = function(callback) {
    var connection = new Connection(config);
    connection.on('connect', function(err){
        if(err){
            callback(err);
        }
        else {
            var request = new Request(
                "SELECT * FROM users" ,
                function(err, rowCount, rows){
                    callback(err, rowCount, rows);
                }
```

```
            );
            connection.execSql(request);
        }
    })
};

module.exports = {
    createUsers: createUsers,
    queryUsers: queryUsers
}
```

15. Update “routes\users.js” file to use following code:

```
var express = require('express');
var router = express.Router();
var db = require('../data/userdb')

/* GET users listing. */
router.get('/', function(req, res, next) {
  db.queryUsers(function(err, rowCount, rows){
    if(err) return next(err);
    res.send(rows);
  })
});

router.put('/', function(req, res, next){
  db.createUsers(function(err, rowCount){
    if(err) return next(err);
    res.send('Added ${rowCount} records');
  });
});

module.exports = router;
```

16. Start application locally by “*npm start*” command and execute http://localhost:3000/users from web browser, website doesn’t return any result because list with users is empty,

17. Tu execute PUT method invoke: “curl -Method PUT -Uri http://localhost:3000/users” from the VS terminal,

18. Refresh http://localhost:3000/users URL and check if there is any raw data presented in the json format,

19. Update stage version of application in Azure, verify if data is available from there as well,