

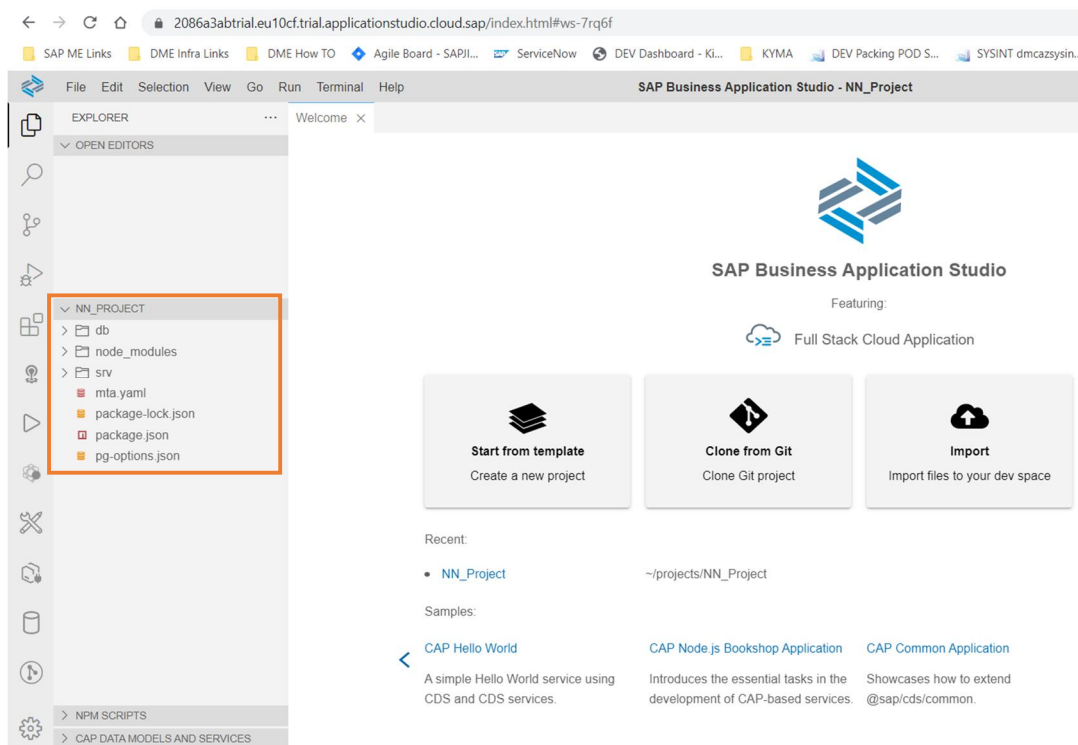
Solution Details

In this tutorial we will look at Next Number extensibility solution provided in the DMC_NextNumber_InAppExtensions/batch-nn-postgresql/code_solution directory in NN_Project.zip archive. We create a full-stack application with Node.js backend bound to PostgreSQL database service that is deployed to Cloud Foundry. PostgreSQL database is used as a database layer to store generated sequence numbers to a custom NN_SEQUENCE table.

Prerequisites

1. Completed all steps from InstallationAndConfigurationGuide.docx document.
2. Run SAP Business Application Studio.

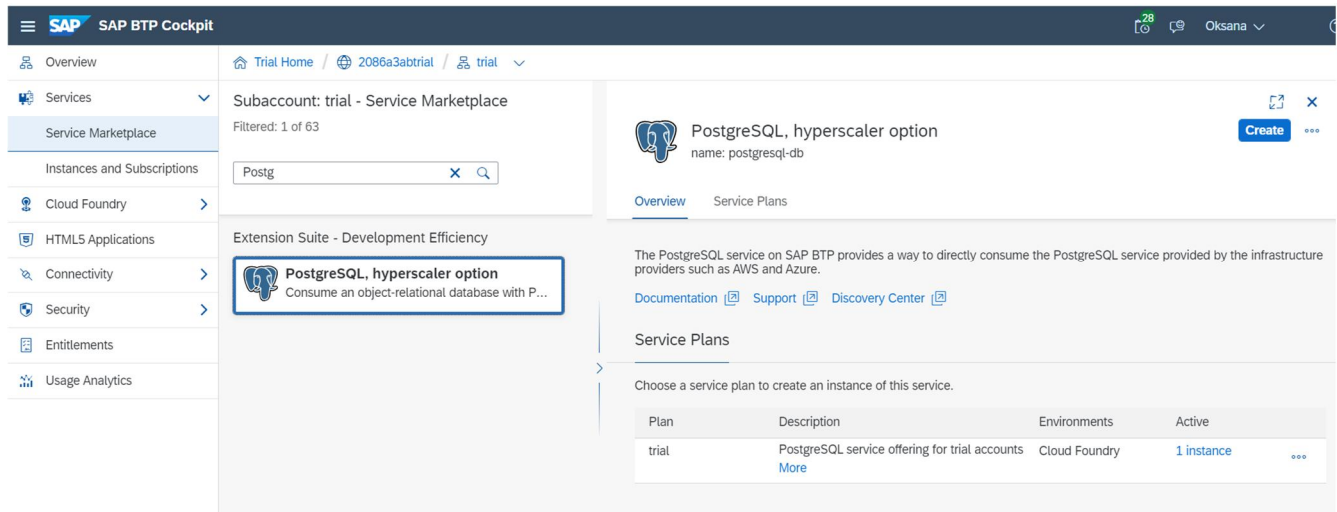
NN_Project is Node.js application that shows how to extend Next Number functionality with a simple REST API solution implemented in SAP Business Application Studio. This Node.js application will be deployed to Cloud Foundry and exposed as a REST API.



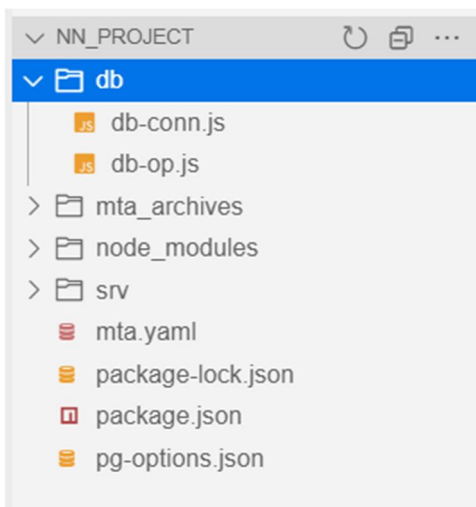
Implementation Details

- Database Layer

PostgreSQL is an object-relational database and was chosen for the database layer. Cloud infrastructure provides the “trial” service plan in this example. PostgreSQL database is used to store the generated sequence number in the custom NN_SEQUENCE table.



NN_Project has “db” folder that contains two JS files to serve database layer needs.



- db-conn.js – it creates NN_SEQUENCE table if doesn't exist yet. Opens connection to PostgreSQL database by using VCAP_SERVICES environment variable.

Script to create NN_SEQUENCE table if doesn't exist yet:

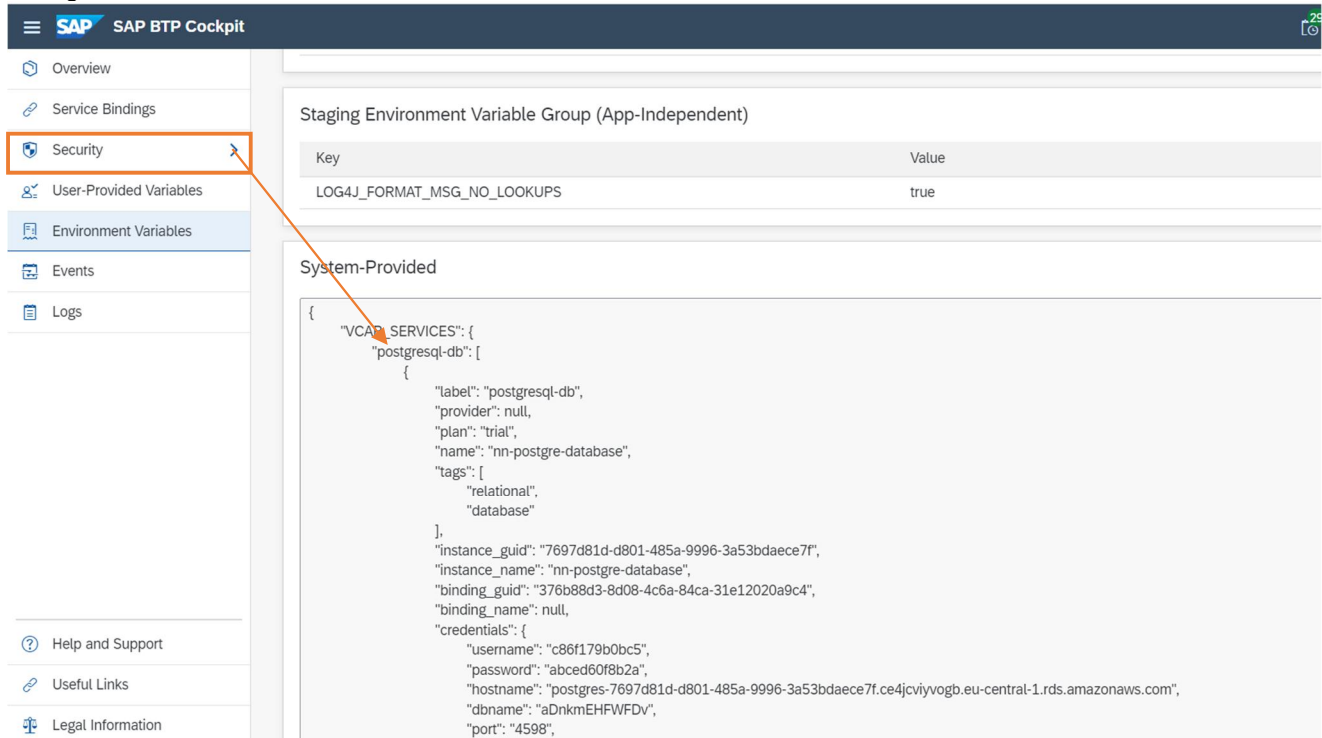
```
const createTable =  
  'CREATE TABLE IF NOT EXISTS NN_SEQUENCE \  
    ( \  
      ID serial PRIMARY KEY, \  
      PLANT VARCHAR , \  
      WORK_CENTER VARCHAR , \  
      RESET_MODE VARCHAR , \  
      DAY INTEGER DEFAULT 0, \  
    )
```

```

MONTH INTEGER DEFAULT 0, \
YEAR INTEGER DEFAULT 0, \
NN_SEQUENCE INTEGER DEFAULT 0, \
unique(PLANT, WORK_CENTER, RESET_MODE, DAY, MONTH, YEAR ) \
);

```

Environment Variables contains VCAP_SERVICES with credentials that can be used to connect to PostgreSQL database.



The screenshot shows the SAP BTP Cockpit interface. In the left-hand navigation menu, the 'Security' option is highlighted with an orange rectangle, and an orange arrow points from it to the 'System-Provided' section of the main content area. The 'System-Provided' section displays a JSON configuration for 'VCAP_SERVICES', specifically for a 'postgresql-db' service. The JSON includes fields for labels, provider, plan, name, tags, instance GUID, instance name, binding GUID, binding name, and a 'credentials' object containing username, password, hostname, dbname, and port.

```

if (process.env.VCAP_SERVICES) {
  // build connection details
  connectionConf = {
    host: xserv.cfServiceCredentials('nn-postgre-database').hostname,
    port: xserv.cfServiceCredentials('nn-postgre-database').port,
    database: xserv.cfServiceCredentials('nn-postgre-database').dbname,
    user: xserv.cfServiceCredentials('nn-postgre-database').username,
    password: xserv.cfServiceCredentials('nn-postgre-database').password,
    ssl: {
      ssl mode: 'require',
      rejectUnauthorized: false
    },
  };
}

```

Use pg-promise to open connection and create table:

```

const pgp = require('pg-promise')();

//open connection to database
const db = pgp(connectionConf);
//console.log("connection configuration: " + JSON.stringify(connectionConf));
// create NN_SEQUENCE table
db.query(createTable)
  .then(function () {

```

```

        //console.log('DB initialized: ' + db);
        cb(null, db);
        return;
    })
    .catch((err) => {
        console.error("DB not initialized - ERROR: " + err);
        cb(err, null);
        return;
    });

```

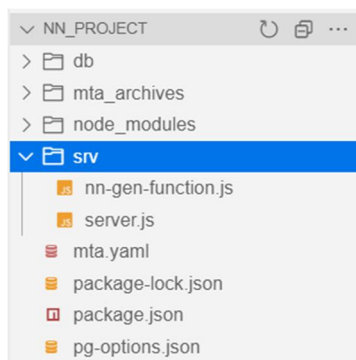
- db-op.js – exposes methods that technically execute SQL statements, such as SELECT, UPDATE, INSERT, DELETE with NN_SEQUENCE table.

- Ø function getAll(db, res) – select all records from NN_SEQUENCE table
- Ø async function upsert(db, upsertData) – insert new or update existing record in NN_SEQUENCE table
- Ø function updateSequence(db, res, id, sequence) – update NN_SEQUENCE column in NN_SEQUENCE table by ID.
- Ø function deleteOne(db, res, id) – delete single record from NN_SEQUENCE table by ID.

• Backend Layer

The server-side of the application is written in Node.js utilizing Express service framework. An Express application is most often used as a backend application in a client-server architecture. Express is a perfect choice for a server when it comes to creating and exposing REST APIs.

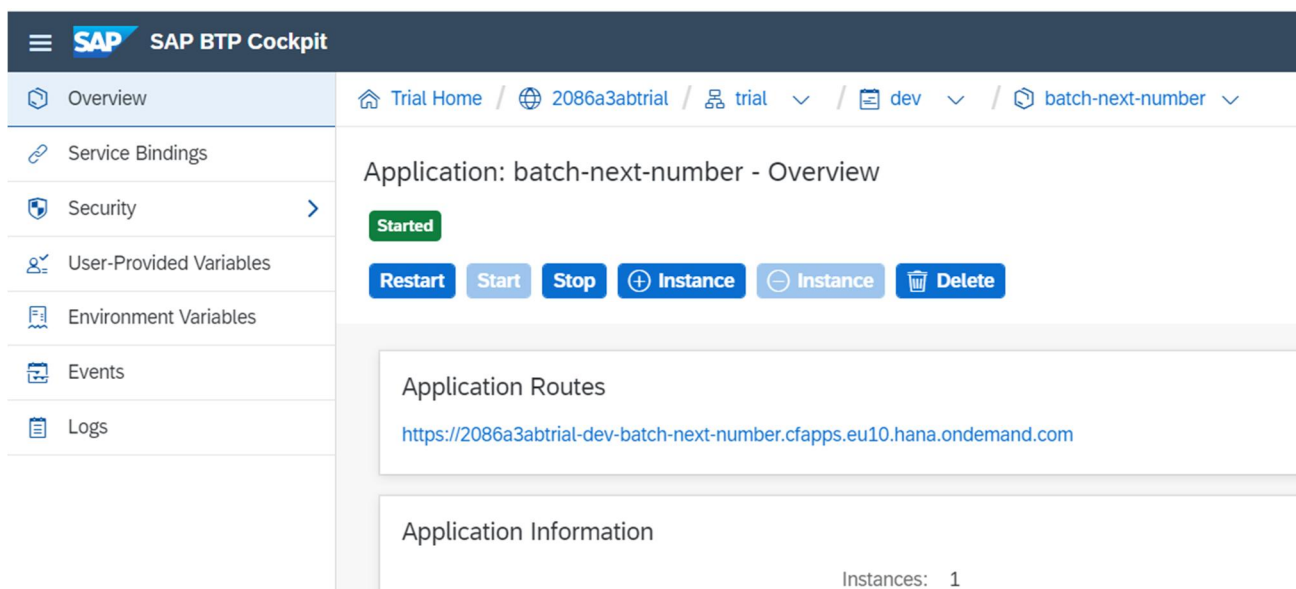
NN_Project has “srv” folder that contains two JS files.



- server.js – used Express for Node.js to build REST API to expose the following operations:
 - Ø GET HTTP method – returns the message “Generate Next Number method GET - deployed!”. Was implemented to verify that deployment to Cloud Foundry finished successfully.
Example: <https://XXXXabtrial-dev-batch-next-number.cfapps.eu10.hana.ondemand.com/>
 - Ø POST HTTP method – executes generateNextNumber method and returns new identifiers in response.
Example: <https://XXXXabtrial-dev-batch-next-number.cfapps.eu10.hana.ondemand.com/>
 - Ø GET HTTP method for /readAll URI – returns all records from NN_SEQUENCE table
Example: <https://XXXXabtrial-dev-batch-next-number.cfapps.eu10.hana.ondemand.com/readAll>

- Ø DELETE HTTP method for /delete/:id URI with "id" parameter – deletes the record from NN_SEQUENCE table by ID
Example: <https://XXXXabtrial-dev-batch-next-number.cfapps.eu10.hana.ondemand.com/delete/7>
- Ø PUT HTTP method for /updateSequence/:id&:sequence URI with "id" and "sequence" parameter – updates "sequence" in NN_SEQUENCE table by ID
Example: <https://XXXXabtrial-dev-batch-next-number.cfapps.eu10.hana.ondemand.com/updateSequence/9&10>

Application routes for HTTP methods above can be found on the Overview screen for the deployed application.



- nn-gen-function.js – builds new identifier in generateNextNumber method and returns new identifiers in response. The method is exposed and called from POST method in srv/server.js

```
// POST method to process request from generate new identifier for next number functionality
app.post('/', async function (req, res) {
  nnGenFun.generateNextNumber(_db, req, res);
});
```

• Configuration Files

- mta.yaml - multitarget application (MTA) development descriptor. This file contains the instructions on how to construct our Cloud application.

"properties" defined in mta.yaml are available after deployment and can be changed at User-Provided Variables.

properties: #module properties for CF Apps can be consumed as app environment variables at runtime
PATTERN: "PLANTYYYYDDMMLLNNNNN"

NUMBER_BASE: "10"
RESET_MODE: "NONE"

The screenshot shows the SAP BTP Cockpit interface. The left sidebar contains navigation links: Overview, Service Bindings, Security, User-Provided Variables (selected), Environment Variables, Events, and Logs. The main area is titled 'Application: batch-next-number - User-Provided Variables' and shows 'All: 4' variables. A table lists the variables:

Key	Value	Actions
DEPLOY_ATTRIBUTES	{ "app-content-digest": "7D7AE057E6037BE21F8440C9CAA29A39" }	[Edit] [Delete]
NUMBER_BASE	10	[Edit] [Delete]
PATTERN	PLANTYYYYDDMMLLNNNNN	[Edit] [Delete]
RESET_MODE	NONE	[Edit] [Delete]

"resources" is used to bind the PostgreSQL database with our application. The service plan is "trial" in our example.

"service-plan" should be changed if you have another service plan provided by the Cloud Foundry environment.

resources:

- name: nn-postgre-database
- parameters:
- path: ./pg-options.json
 - service: postgresql-db
 - service-plan: trial
 - skip-service-updates:
 - parameters: true
- type: org.cloudfoundry.managed-service

After deployment you can check Service Bindings for your application

The screenshot shows the SAP BTP Cockpit interface. The left sidebar contains navigation links: Overview, Service Bindings (selected), Security, User-Provided Variables, Environment Variables, Events, and Logs. The main area is titled 'Application: batch-next-number - Service Bindings' and shows 'All: 1' binding. A table lists the service binding:

Name	Service	Plan	Actions
nn-postgre-database	postgresql-db	trial	[Edit] [Delete]

- package.json - holds important information about the project. It contains human-readable metadata about the project (like the project name and description) as well as functional metadata like the package version number and a list of dependencies required by the application.

In scripts section you can define script commands that can build and deploy application to Cloud Foundry, here is example:

```
"scripts": {  
  "start": "node srv/server.js",  
  "build:cf": "mbt build",  
  "deploy:cf": "cf deploy mta_archives/NN_Project_1.0.0.mtar"  
}
```

Run such scripts you should execute in terminal: `npm run <script name>`