



App Platform

Step by Step

Building a simple but complete application with Node.js

<https://www.digitalocean.com>

What we are going to do today?

- Develop the **front-end**
 - Deployed a static site
- Implement a **back-end API**
 - Provision a node backed
- Add persistence with a **database**
 - Provisioning a Postgresql Databse
- Develop freely with continuous updates!

Configuration

Preparation

Register for a free account

- You will get access to Atlantis

Install the cli `doctl`

<https://docs.digitalocean.com/reference/doctl/how-to/install/>

- Available for Windows, Mac and Linux

Get your token

- Install the `doctl` cli
- Click on **API** on Menu
- Click on **Generate New Token**
- Click on **Copy**
- Type `doctl auth init` and paste

The screenshot shows the DigitalOcean API console interface. The left sidebar contains a menu with items like 'ops', 'NEW', 'proplets', 'ubernetes', 'olumes', 'atabases', 'paces', 'ontainer Registry', 'nages', 'etworking', 'onitoring', 'ettings', 'lling', and 'API'. The main content area is titled 'Applications & API' and has tabs for 'Tokens/Keys', 'OAuth Applications', and 'Access'. Under 'Tokens/Keys', there's a section for 'Personal access tokens' with a 'Generate New Token' button. A modal window titled 'New personal access token' is open, showing a 'Token name' field with 'macbookari', 'Select scopes' with 'Read (default)' and 'Write (optional)' checked, and a 'Generate Token' button. Below the modal, a table shows the generated token details.

Name	Scope
macbookari	READ WRITE
377c7c0c161210369879e2327e5fd50b5681cad864cde6b2df3082a951e0a2a1 Copy	

MANAGE

Apps **NEW**

Droplets

Kubernetes

Volumes

Databases

Spaces

Container Registry

Images

Networking

Monitoring

Settings

Billing

API

Applications & API

Tokens/Keys OAuth Applications Access

Personal access tokens

Tokens you have generated to access the [DigitalOcean API](#).

Name	Scope	Created
msciabarra		

Generate New Token

New personal access token

Token name

Enter token name
macbookari

Select scopes

☒ Read (default)

☒ Write (optional)

Read our [personal access token documentation](#) for more information on scopes.

Generate Token

Name

Scope

macbookari

READ

WRITE

377c7c0c161210369879e2327e5fd50b5681cad864cde6b2df3082a951e0a2a1 Co

4

Frontend

Creating a frontend

- An example using Svelte

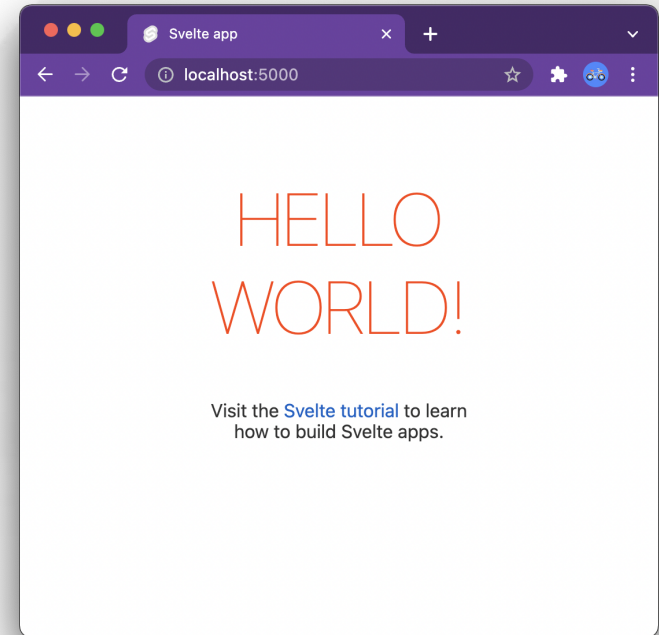
```
npx degit  
sveltejs/template frontend
```

- Install it

```
cd frontend && npm install
```

- Run in development mode

```
npm run dev
```



Concepts of App Platform / 1

- You deployment is the `.do/app.yaml`
- It includes lots of components:
 - static sites
 - applications in multiple languages
 - databases
 - and much more...

Concepts of App Platform / 2

- The YAML describes the complete cycle:
- **1** Pulling from repositoryes
- **2** Building applications
- **3** Exposing to the internet

Deployment

- Deploy with: `doctl app create --spec .do/app.yaml`

Deployment: `.do/app.yaml`

```
name: tutorial-app-platform
static_sites:
- name: frontend
  # 1 pulling from repositories
  github:
    repo: sciabarrado/tutorial-app-platform
    branch: main
    deploy_on_push: true
  # 2 building applications
  build_command: npm run build
  source_dir: frontend
  # 3 exposing to the internet
  routes:
  - path: /
```

Exercise: deploy frontend

```
# creating the frontend
npx degit sveltejs/template frontend
cd frontend && npm install
npm run dev
# deploying the frontend
mkdir .do
cp src/1-app.yaml .do/app.yaml
doctl app create --spec .do/app.yaml
# monitoring
ID=$(doctl app list | awk '/tutorial-app-platform/ { print $1}')
echo $ID
doctl app logs $ID
```

Backend

Let's build our backend

- We are going to use **Node.js**
 - you can use also out of the box Python, PHP, Golang, Ruby
 - You can also use "*whatever*" thanks to Dockerfile
 - you need a bit more knowledge here
- Builds are automated thanks to "buildpack"
 - they can build your code *automagically* in many common cases

Simple Backend Code `node.js`

```
const express = require('express')
const app = express()
const port = 8080

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`App listening at http://localhost:${port}`)
})
```

Creating the backend

```
# a new directory for backend  
mkdir backend  
cd backend  
# mandatory initializations  
npm -y init  
npm install --save express  
# using our examples here  
cp ../src/2-index.js index.js  
node index.js
```


Deploying the backed with `app.yaml`

- Adding a `services` section
- same steps as before:
 - **1** pull
 - **2** build (automated)
 - **3** expose to interned
- Additional step:
 - **4** run your code

Backend deployment

```
services:  
- name: backend  
  # 1 pull  
  github:  
    repo: sciabarrado/tutorial-app-platform  
    branch: main  
    deploy_on_push: true  
  source_dir: backend  
  # 2 build is autodetected  
  # 3 expose to interned  
  routes:  
  - path: /api  
  # 4 run your code  
  run_command: node index.js
```

Exercise: backend

```
# new configuration
cd ..
cp src/2-app.yaml .do/app.yaml
# update
ID=$(doctl app list | awk '/tutorial-app-platform/ { print $1}')
echo $ID
doctl app update $ID --spec .do/app.yaml
```

Database

What you need to know about the database

- It is automated provisioned:
 - just add it to the `app.yaml`
- Available:
 - SQL: `postgresql`, `mysql`
 - NoSQL: `redis`, `mongodb`
- You need to use environment variables to connect to it

Environment variables used with PostgreSQL

- `PGHOST`, `PGPORT`
 - hostname and port of the database
- `PGUSER`, `PGPASSWORD`
 - username and password
- `PGDATABASE`, `PGSSLMODE`
 - database name
 - *important* you may need to set SSL mode in certain cases

Exercise: create database locally

```
# create the user
psql postgres -U postgres
create user demo with password 'demo';
create database localdb with owner = 'demo';
quit
# configure the environment variables
export PGHOST=localhost
export PGPORT=5432
export PGDATABASE=localdb
export PGUSER=demo
export PGPASSWORD=demo
# check the connection
psql -h $PGHOST -p $PGPORT -U $PGUSER $PGDATABASE
```

Connect to database with node

```
# install driver
cd backend
npm install pg --save
node
# test database connection
const { Client } = require('pg')
const client = new Client()
await client.connect()
let create = `
CREATE TABLE IF NOT EXISTS guestbook(
  id SERIAL PRIMARY KEY,
  message TEXT)`
const res = await client.query(create)
```


Connecting to the database

```
const { Client } = require('pg')

function start() {
  console.log("connecting to database")
  let client = new Client()
  client.connect()
    .then(() => init(client))
    .catch((err) => {
      console.log(err)
      setTimeout(start, 2000)
    })
}
```

Initializing the database

```
let create = `
CREATE TABLE IF NOT EXISTS guestbook(
  id SERIAL PRIMARY KEY,
  message TEXT
)`

function init(client) {
  client.query(create)
    .then(() => app.listen(port))
    .then(() => console.log(`App listening a at ${port}`))
    .catch(console.log)
}
```

Provisioning a database

```
databases:  
- name: db  
  engine: PG  
  version: "12"
```

- Development database
 - do not use in production

Connecting to the database

```
# add parameters to connect to database
```

```
envs:
```

- name: PGHOST
value: \${db.HOSTNAME}
- name: PGPORT
scope: RUN_TIME
value: \${db.PORT}
- name: PGDATABASE
scope: RUN_TIME
value: \${db.DATABASE}
- name: PGUSER
scope: RUN_TIME
value: \${db.USERNAME}
- name: PGPASSWORD
scope: RUN_TIME
value: \${db.PASSWORD}

Exercise: deployment database

```
cp src/3-app.yaml .do/app.yaml
cp src/3-index.js backend/index.js
# update
ID=$(doctl app list | awk '/tutorial-app-platform/ { print $1}')
echo $ID
doctl app update $ID --spec .do/app.yaml
```

Guestbook