Creating a Serverless Function from the Command Line

(note these are all for a macOS) (Made for Digital Ocean specific services)

Step 1: Using doctl

Importance: doctl is the package that helps connect your command line/computer to digital ocean

Install doctl using this code in your terminal

```
Unset
brew install doctl
```

Create your own personal API token in the DigitalOcean web interface. This will be under **Applications & API.** Make sure you save your API code because it only shows you once. You will use this token to connect to DigitalOcean using doctl.

```
Unset
doctl auth init
```

Once you enter the code above the terminal will prompt you to enter your API token.

To confirm your account is linked, put this in the command line and it should output your account details.

```
Unset
doctl account get
```

Now, link the serverless function application to your computer

```
Unset
doctl serverless install
```

Ensure installation

```
Unset
doctl serverless status
```

Step 2: Create a function namespace

Importance: this is the namespace where your serverless function will live. This ensures organization.
From the command line:

```
Unset
doctl serverless namespaces create --label "example-namespace"
--region "nyc1"
```

After label you can change the name, for example “ASPA-WUDR” and choose your region. A list of regions can be found here. You can also use the command line: `doctl serverless namespaces list-regions`

If the namespace is correct, a connected message will be the response to the namespace function.

Step 3: Create Function

Using the serverless package of doctl, we will initialize a new “project” (directory name) and select a runtime/language your function will operate in.

```
Unset
doctl serverless init --language js example-project
```

For python you use “py”. If correct you will get a response that says the function/project was created.

The function created has a default project structure. This project structure has now been saved to your local machine.

Step 4: Editing Function

Now we are going to edit the function we have created. Navigate to your project directory (the name you just gave it.

```
Unset
cd example-project
```

And list the contents of the directory

```
Unset
ls
```

The output should be your project structure which is
Packages Project.YML

The project structure is extremely embedded. The default python file Digital Ocean saves is called “hello.py”. Here is an example of all the directories you can open and list

(base) Lizas-MacBook-Pro:~ lizamclatchy$ cd data_cleaning
(base) Lizas-MacBook-Pro:data_cleaning lizamclatchy$ ls
packages project.yml
(base) Lizas-MacBook-Pro:data_cleaning lizamclatchy$ cd packages
(base) Lizas-MacBook-Pro:packages lizamclatchy$ ls
sample
(base) Lizas-MacBook-Pro:packages lizamclatchy$ cd sample
(base) Lizas-MacBook-Pro:sample lizamclatchy$ ls
hello
(base) Lizas-MacBook-Pro:sample lizamclatchy$ cd hello
(base) Lizas-MacBook-Pro:hello lizamclatchy$ ls
hello.py

“hello.py” is where you want to put your code for the function. You can edit the hello.py code in the terminal.

```
Unset
nano hello.py
```

To close and save your .py file press `Ctrl + X` to exit.

- It will prompt you to save changes; press `Y` for yes.
- Press `Enter` to confirm the file name.

If you want to rename the .py, use this command

```
Unset
mv hello.py main.py
```

Step 5: Function Specifics

Importance: extras you need to run your function from the command line
Everything project structures (.yml) and requirements.txt and build.sh

Example project structure:

```
Unset

example-project
├── packages
│   ├── sample
│   │   ├── hello
│   │   │   ├── main.py
│   │   │   └── requirements.txt
│   │   └── build.sh
└── project.yml
```

Project.yml :
Store -> Next to your packages folder NOT IN IT.
Purpose ->This tells digital ocean what to expect in your project structure. Goal = keep it easy and simple.
NOTE: Sample and hello are default folder names. You can change them but you also have to change them in your .yml file

```
packages:
  - name: sample
    actions:
      - name: hello
        runtime: 'python:3.11'
```

Requirements.txt
Store -> in the same folder as your main.py script
Purpose -> Put your packages and package versions in this .txt file
There are a list of packages already imported in digital ocean in the requirements hyperlink

Example: create in your text editor

```
Unset
py.jokes==0.6.0
```

build.sh
Store -> in same folder with requirements.txt and main.py
Purpose -> loads your package into a virtual environments

A default prompt:Can save as a .txt file and just change the . extension to .sh

```
Unset
set -e

virtualenv --without-pip virtualenv
pip install -r requirements.txt --target
virtualenv/lib/python3.9/site-packages
```

Only edit - your runtime environment

Step 6: Deploying/Running Your Function

Up to this point:

- You have your code inputted and everything named how you want it
- Your files are all in correct places

Navigate to your directory

```
Unset
cd data_cleaning
```

Deploy your function (NOTE THIS DEPLOYS IT TO THE DIGITAL OCEAN INTERFACE IT DOES NOT RUN IT)

```
Unset
doctl serverless deploy
```

Your function has now been deployed to your digital ocean namespace. This is where you can add necessary environmental variables, update memories/runtimes, and other.

Run your function in the Digital Ocean interface

My favorite resource:
https://github.com/digitalocean/sample-functions-python-jokes/blob/master/README.md

You can clone this repo and follow the prompts if you need a starting point (I highly recommend this as this is what I did)