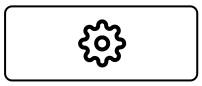
Starting from Scratch: Basic Web Application

We'll cover the following

In this part, we will walk you through getting a basic web application running in the cloud on AWS. We will start with a blank project, and build the application and its infrastructure step by step. Each step focuses on a single aspect of the infrastructure, and we will try to explain in detail what's happening, and why.



All the source code shown in this guide is available on GitHub. Each commit represents a code checkpoint from the book, and you can find it all here: AWS Bootstrap (https://github.com/good-parts/aws-bootstrap).

When interacting with AWS, we will use both the AWS console (https://console.aws.amazon.com) and the AWS CLI (https://aws.amazon.com/cli). In addition, we will also make use of the following tools:

• GitHub (https://github.com) as the source code repository for our application and infrastructure code.

• node.js (https://nodejs.org) and npm (https://www.npmjs.com) to build our application.

• git (https://git-scm.com/) for version control.

• curl (https://curl.haxx.se/docs/manual.html) to interact with our application.

Objective#

- Get a simple web application running on a single EC2 instance.

Steps#

- Write a basic “hello world” web application.

In this section, we will create a tiny web application and we will get it running on an EC2 instance in an AWS account. We will start by

performing all the steps manually, and we will automate them in later

sections. Our application is not very interesting, and the way it will be hosted is far from ideal, but it will allow us to spend the rest of this book building a well-contained AWS setup, step by step.

Creating our application#

> We will need `git` and `npm` installed. But don't worry, these are already installed on our platform. So you can straight away use them in our terminal widget.

Now, let's create our bare-bones application, along with a git repository to store it.

```
mkdir aws-bootstrap && cd aws-bootstrap
git init
npm init -y
```

terminal

Our application is going to listen for HTTP requests on port 8080 and respond with "Hello World". The entire application will be in one file, `server.js`.

```
const { hostname } = require('os');
const http = require('http');
const message = 'Hello World\n';
const port = 8080;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(message);
});
  server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname()}:${port}/`);
```

```
});
```

server.js

> ⚠️ **Line #4:** we'll run the server on port `8080` because port numbers below `1024` require root privileges.

We can run our application directly with the `node` command.

```
node server.js
Server running at http://localhost:8080/
```

terminal

And we can test it with `curl` from another terminal window.

```
curl localhost:8080
Hello World
```

terminal

All these commands are already executed in the below widget. Hit on the RUN button to start the server.

```
const { hostname } = require('os');
const http = require('http');
const message = 'Hello World\n';
const port = 8080;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(message);
});
```

```
  server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname()}:${port}/`);
});
```

Next, let's use a process manager to monitor our application so that it automatically restarts if it crashes. To do this, we need to modify our `package.json` file.

```
{
  "name": "aws-bootstrap",
      "version": "1.0.0",
      "description": "",
      "main": "server.js",
      "scripts": {
  "start": "node ./node_modules/pm2/bin/pm2 start ./server.js --name hello_aws -
  "stop": "node ./node_modules/pm2/bin/pm2 stop hello_aws",
  "build": "echo 'Building...'"  },
  "dependencies": {
    "pm2": "^4.2.0"
  }
}
```

package.json

Line #11: Takes a dependency on pm2, a node process manager.

Line #7: From now on, we'll use npm start to start our application via pm2.

Line #7: pm2 will monitor it under the name hello_aws and send its stdout to `../logs/app.log`

Line #8: We'll use npm stop to tell pm2 to stop our application.

Line #9: A dummy build step. Your actual production build process goes here.

Now, we need to use npm to get the new dependency we added in `package.json`.

```
npm install
```

terminal

In `package.json` we specified that the application's logs are sent to `../logs`. Having the directory for logs outside the application's directory will be important when we deploy this with CodeDeploy, since it prevents the logs directory from being deleted with every deployment. For now, let's create the directory manually on our local machine.

```
mkdir ../logs
```

terminal

And now we should be able to start our application through the process manager.

```
npm start

[PM2] Applying action restartProcessId on app [hello_aws](ids: [ 0 ])
[PM2] [hello_aws](0) ✓
[PM2] Process successfully started
┌─────┬─────────────┬─────────┬─────┬──────────┬──────────┬──────────┐
│ id  │ name        │ mode    │ ↺   │ status   │ cpu      │ memory   │
├─────┼─────────────┼─────────┼─────┼──────────┼──────────┼──────────┤
│ 0   │ hello_aws   │ fork    │ 0   │ online   │ 0%       │ 8.7mb    │
```

terminal

Testing via `curl` should once again show our "Hello World" message.

```
curl localhost:8080
Hello World
```

terminal

Again, all these commands are already executed in the widget below. Hit on the RUN button to start the server.

```
const { hostname } = require('os');
const http = require('http');
const message = 'Hello World\n';
const port = 8080;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(message);
});
  server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname()}:${port}/`);
});
```

And with everything working, we can now commit all our changes to git.

```
git add server.js package.json package-lock.json
git commit -m "Create basic hello world web application"
```

terminal

Pushing our code to GitHub#

If you don’t already have a GitHub account, create one (https://github.com) (it’s free for simple projects like this).

Next, create a new GitHub repository (https://github.com/new). We named ours `aws-bootstrap`. The repository needs to be public for now, but we’ll be able to make it private once we set up CodeBuild.

Now that we have a GitHub repository, let’s push our local changes to GitHub.

```
git remote add origin https://github.com/<username>/aws-boostrap.git
git push -u origin master
```

terminal

Line #1: replace `<username>` with your GitHub username.

Again all these git commands are executed. You just need to set your Github username as the *environment variable* first.

> **NOTE:** When you run the code, it will ask you your *username* and *password* before pushing the files to Github. **So don't miss that!**

```
const { hostname } = require('os');
const http = require('http');
const message = 'Hello World\n';
const port = 8080;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(message);
});
  server.listen(port, hostname, () => {
    console.log(`Server running at http://${hostname()}:${port}/`);
});
```

In the next lesson, we will host our application and manually install it on an EC2 instance.

← Back

Next →

Quiz - Basic AWS Services

Starting from Scratch: Manual AWS In

Quiz – Basic AWS Services Starting from Scratch: Manual AWS In...