How would a software developer create technical documents and sell them using the disruptive power of Web3 technologies? (Part I)

Author: Paulo Jerônimo - https://paulojeronimo.com
Last update: 2023-06-09 10:50:37 -0300

Online PDF version:
https://paulojeronimo.com/asciidoctor-web3-article/private/sample.pdf

Follow me, and give feedback about this article, here:
https://www.linkedin.com/feed/update/urn:li:activity:7068879548234313728/

If you write technical texts (like articles, tutorials, ebooks, etc) do you know that with a single text writing in AsciiDoc format (like this one) and the right tools, you could offer more value to your readers (customers) by presenting it in different formats?

Did you know that if you acquire knowledge in Web3, you will have much more creative ways to sell your texts?

In this article, I will explore a little more about this subject, beginning by showing you how to generate many outputs from one unique document code (this).

In Part II of this article (coming soon), I will detail how Web3 technologies can achieve the second part of the story: help authors sell their texts using innovative ideas as a plus to the traditional means of payment, which involves prominent editors.

Table of Contents

Before we start, let me introduce myself

I (the author of this article) work as a Full Stack Developer who has been acting in several areas of software development (since 1993). I have experience from the infrastructure to the front end, passing through extensive coding time in middleware and the back end.

I'm also a triathlete, three (3) times Ironman.

So I didn't get used to not being an expert in swimming, cycling, or running, just as I don't consider myself, in my work, an infrastructure, backend, or frontend expert.

In the software development world, my role has been to put the pieces of a giant puzzle together. Today, with the help of technologies of Artificial Intelligence, my work has been made easier to do.

That way, some of my software tutorials (like this one) avoid discussing a single topic. Instead, they combine several things, providing a more realistic view of how the software works.

As you can read, **I am a *Technical Writer***. I need to write about things that I develop or learn, and this is a thing that I like to do. You can read many of my texts at the following link:

https://paulojeronimo.com/sitemap/#articles-or-tutorials.

I'm a Brazilian guy. So English is not my primary language. But I am doing my best to share my knowledge with you in this language. At the same time, I intend to help other software developers who speak Portuguese improve their skills. That's why I also write closed captions in Portuguese for my videos.

1. This is a small article in which I insert a tutorial (a little longer) to explain how you could write it to turn your readers into customers!

As a ***Technical Writer***
(and a [FullStackDeveloper]),

I created a tutorial
(**Super easy to follow**, as you will see in a moment! ☺)
You don't need to be a programmer like me
to write good professional tutorials and books.
But **you need to know how to write better!**

This tutorial is written in AsciiDoc format
(using Asciidoctor.js,
one extension I developed for it,
and other charming tools)
Here is the source code of it:
https://paulojeronimo.com/asciidoctor-web3-article/private/sample.adoc
(this was the original code I wrote)

If you want to be a Full Stack Developer
(starting from using AWS and Java, for example)
you may want to start your journey by following this tutorial.
But ... wait ... In fact ...
this is not a tutorial for newbie programmers!

This is an article that I wrote for *Technical Writers*!

So, **the exciting part** of this text
is not about the following tutorial!
The cool here is the format (in the sample.adoc) presented by
it, not the content.
It's about the way you can write tutorials with the structure and the
right tools to benefit you and your customers.
It's about the benefits you can give to your
readers (when they buy this tutorial).

2. A sample tutorial that you, as a *Technical Writer*, could take as an example to write yours (pay attention to its format!)

As mentioned, pay attention to the format I wrote in (sample.adoc). At some point, I will launch my course covering it and other tools.

I wrote this tutorial with an audience studying to take the AWS Certified Developer certification exam.

Intending to sell texts related to this theme, my strategy to sell them is, among others, to present the execution of the private version of my tutorial in a video on my YouTube channel. But, in the video description, I offer a link to the public text of my tutorial and show the benefits readers will get when they buy it.

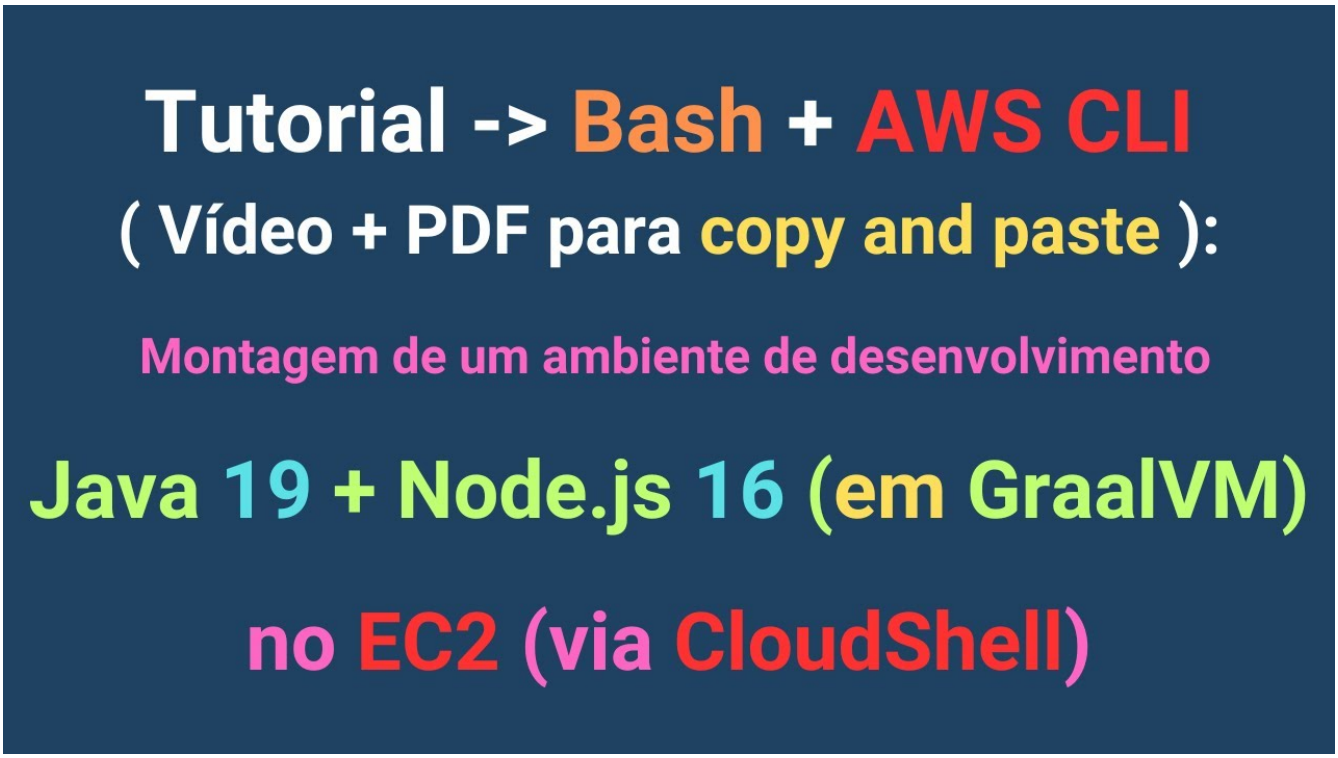
So, in short, to sell my tutorials, my strategy needs:

1. A public version of my tutorial.
    a. This link above is copied to the video description where I it.
    b. It includes a section where I present my benefits to buyers.
2. A private version of it, including this PDF you are reading.
    a. It includes a section where I thank my customers and give them all their benefits.

2.1. Building your Linux development environment on AWS (paid version)

2.1.1. Introduction

Video recorded in Portuguese.
Click the image below to open it on YouTube.



In practical terms, you will not like creating an AWS environment using what I will present in this tutorial. To create a cloud infrastructure on AWS, solutions like AWS CDK, Terraform, or even Vagrant (the way I showed in this Git repository (which I built in 2018)) are much more practical and straightforward.

I also use many other alternatives (cheaper and very fast to mount) to create development environments. For example, AWS Cloud9, GitHub Codespace, Gitpod, etc.

However, the basic knowledge (of AWS and Java fundamentals) I will give you in this tutorial is quite valuable for learning purposes. For example, suppose you want to obtain the AWS Certified Developer certification. What you will practice using the commands below will serve as a basis for understanding IAM and EC2, which, in turn, are required in the test for this certification.

So, in the first part, I demonstrate how to create a machine to develop applitions using Java and Node.js. In part 2, I illustrate the use of this environment to construct charming applications in my following tutorials.

In this tutorial my focus is not teatch you how to develop in Java or Node.js. So, here I'll show you only a simple "Hello World!" in both languages.

Finally, I present how you can unbuild the cloud environment.

2.1.2. What do we need to install to follow this tutorial?

Nothing but a browser!

We'll only need to have access to our root AWS account. So, if you don't have an account on AWS, create it.

After that, we'll be using AWS CloudShell to **copy and paste** all the commands (Bash scripts) in this tutorial to it. These scripts extensively use AWS CLI commands (commented on them).

2.1.3. What do we need to know before starting it?

Only two programming languages, at a basic level of knowledge:

- Shell Scripting (Bash).
- Java. We'll use the GraalVM.
- JavaScript (with Node.js inside GraalVM).

We'll build a Linux development environment. So, I assume you know the basics about it too.

2.1.4. → Part 1 ←

All the commands that I will show you here need only to be typed (or copied) to a terminal running Bash (AWS CloudShell). So, just open it and start to **copy and paste** the following commands. Simple as that! ☺

You can right-click on each code block in image format using this PDF to open it in your browser. Then you can use the famous "**copy and paste**".

Creating an AWS user

```
{
  echo Creating the User
  aws iam create-user --user-name FullStackDev

  echo Creating an Access Key
  access_key=$(aws iam create-access-key --user-name FullStackDev)

  echo Saving the access key
  echo "$access_key" > FullStackDev.access-key.json

  echo Attaching the required policies to the user
  for policy in AmazonEC2FullAccess AWSCloudShellFullAccess
  do
    aws iam attach-user-policy --user-name FullStackDev \
      --policy-arn arn:aws:iam::aws:policy/$policy
  done

  echo -n Retrieving the Account ID '(12 digits): '
  aws sts get-caller-identity --query 'Account' \
    --output text > FullStackDev.accountid.txt
  cat FullStackDev.accountid.txt

  echo Creating a password for the user FullStackDev

  # See the NOTE password-generation
  echo "1^$(tr -cd '[:alnum:]' < /dev/urandom |
    fold -w11 | head -n1)" > FullStackDev.password.txt

  # After the command below, you can test your login
  # (NOTE login-testing)
  echo Creating a login profile for user FullStackDev
  aws iam create-login-profile --user-name FullStackDev \
    --password $(<FullStackDev.password.txt)
}
```

Some NOTES about the code above:

- **password-generation** → The password is randomly generated. It starts with the fixed 1^. Then it gains 11 more characters. This is to avoid the following error:

  ```
  An error occurred (PasswordPolicyViolation) when calling the CreateLoginProfile
  operation: Password should meet 1 more of the following requirements: Password
  should have at least one number, Password should have at least one symbol
  ```

- **login-testing**: Use the output of the next command in order to login on AWS through a new

incognito browser window:

```
cat <<EOF
Open https://$(<FullStackDev.accountid.txt).signin.aws.amazon.com/console

-> Credentials <-
IAM User name: FullStackDev
Password: $(<FullStackDev.password.txt)

-> References ->
AWS/sign-in-urls-defined
EOF
```

After logging, access the AWS CloudShell and try this command:

```
aws iam get-user --query 'User.UserName' --output text
```

You will receive the following error:

```
An error occurred (AccessDenied) when calling the GetUser operation: User:
arn:aws:iam::229248205759:user/FullStackDev is not authorized to perform:
iam:GetUser on resource: user FullStackDev because no identity-based policy
allows the iam:GetUser action
```

So, back back to the cloudshell opened with your root account, type the following command to give that permission to the user `FullStackDev`:

```
aws iam attach-user-policy --user-name FullStackDev \
  --policy-arn arn:aws:iam::aws:policy/IAMReadOnlyAccess
```

Now, test the last `FullStackDev` command. You will see that it will succeed!
Sign out `FullStackDev` and close its incognito window.

Defining some AWS environment variables

The variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` are used to indicate that we will be using the user `FullStackDev` in our following AWS CLI commands.

```
{
  echo Defining some environment variables
  export AWS_ACCESS_KEY_ID=$(jq -r '.AccessKey.AccessKeyId' \
    ~/FullStackDev.access-key.json)
  export AWS_SECRET_ACCESS_KEY=$(jq -r '.AccessKey.SecretAccessKey' \
    ~/FullStackDev.access-key.json)
  export AWS_DEFAULT_REGION=us-east-1
}
```

Instead of using these environment variables, we could also use the command `aws configure` to set them in a directory called `~/.aws/`. This command will create files named `credentials` and `config` inside this directory with the same information configured in the variables above.

1. We are using these variables in the cloud shell we are using for the root user because we want all objects created from the following commands to be associated with the FullStackDev user and not with our root user. That's good practice!
2. We wouldn't need to define these environment variables if we kept the CloudShell window used by the FullStackDev user open and executed the instance creation commands inside it.

Creating the AWS instance

Let's configure some other environment variables related to the machine we'll create and export them:

```
{
  export my_machine_name={my_machine_name}
  export my_machine_security_group=DevMachines
}
```

Create the instance:

```
{
  echo Creating the Security Group \"$my_machine_security_group\"
  aws ec2 create-security-group \
    --group-name $my_machine_security_group --description $my_machine_name

  echo Authorizing access to port 22
  aws ec2 authorize-security-group-ingress \
    --group-name $my_machine_security_group --protocol tcp --port 22 \
    --cidr 0.0.0.0/0

  echo Creating the Key Pair
  aws ec2 create-key-pair --key-name FullStackDev \
    --query 'KeyMaterial' --output text > FullStackDev.pem
  chmod 400 FullStackDev.pem

  echo Building a backup of all credentials. \
    Save it to your computer!
  tar cvf FullStackDev.tar FullStackDev.*

  echo Launching the EC2 Instance
  aws ec2 run-instances --image-id ami-053b0d53c279acc90 \
    --count 1 --instance-type t2.medium --key-name FullStackDev \
    --security-group-ids $my_machine_security_group --tag-specifications \
    "ResourceType=instance,Tags=[{Key=Name,Value=$my_machine_name}]"
}
```

Wait for the instance creation:

```
{
  echo Waiting for \"$my_machine_name\" to Start ...
  aws ec2 wait instance-running --filters \
    "Name=tag:Name,Values=$my_machine_name"

  echo -n 'Retrieving the Instance ID: '
  instance_id=$(aws ec2 describe-instances --filters \
    "Name=tag:Name,Values=$my_machine_name" --query \
    'Reservations[].Instances[].InstanceId' --output text)
  echo $instance_id

  echo -n 'Retrieving the Public DNS Name: '
  public_dns_name=$(aws ec2 describe-instances \
    --instance-ids $instance_id --query \
    'Reservations[].Instances[].PublicDnsName' --output text)
  echo $public_dns_name

  echo "ubuntu@$public_dns_name" > FullStackDev.$my_machine_name
  tar rf FullStackDev.tar FullStackDev.$my_machine_name
}
```

Saving the user credentials

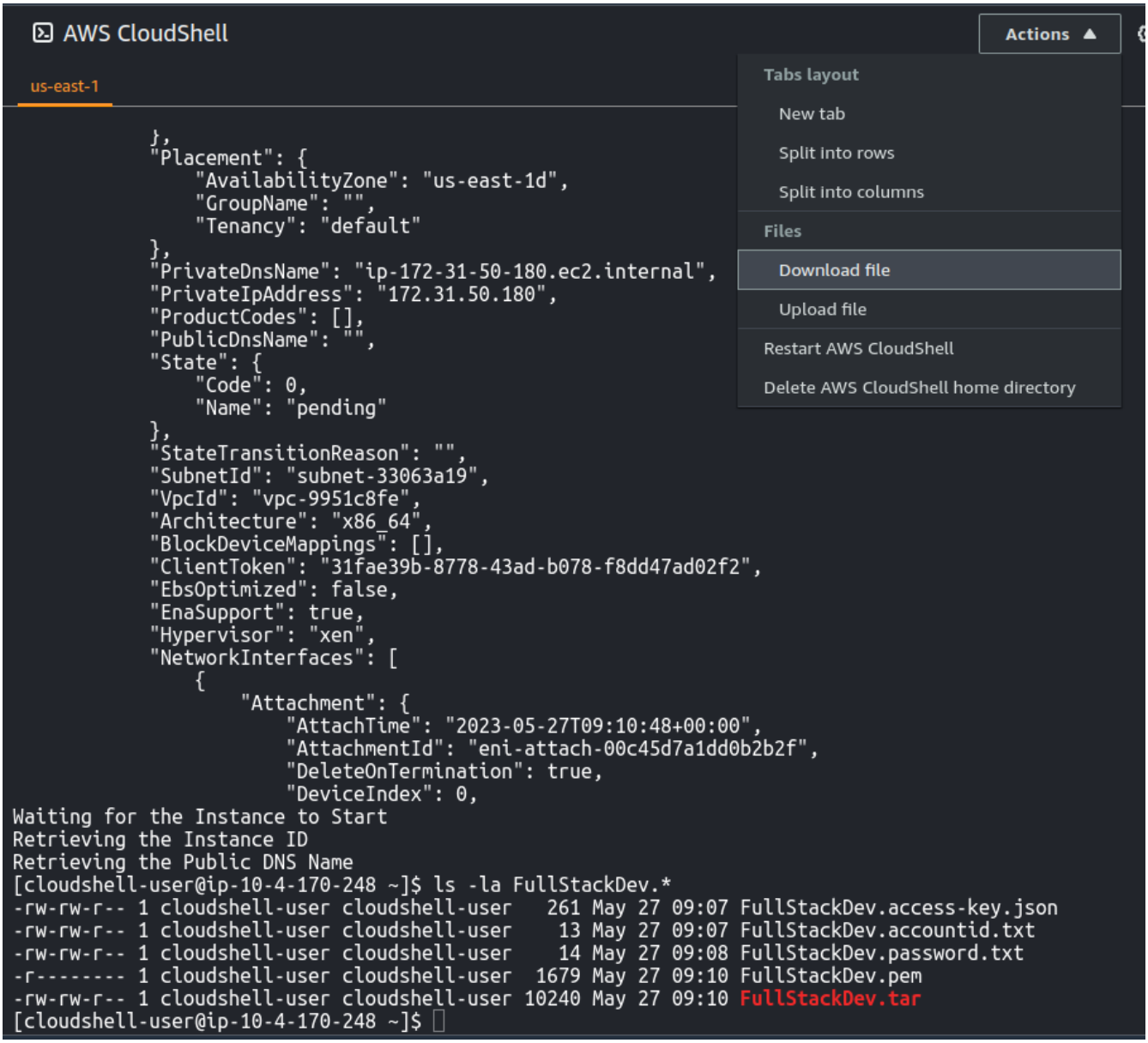
Not that you you will have a bunch of `FullStackDev` files generated:

```
ls -la FullStackDev.*
```

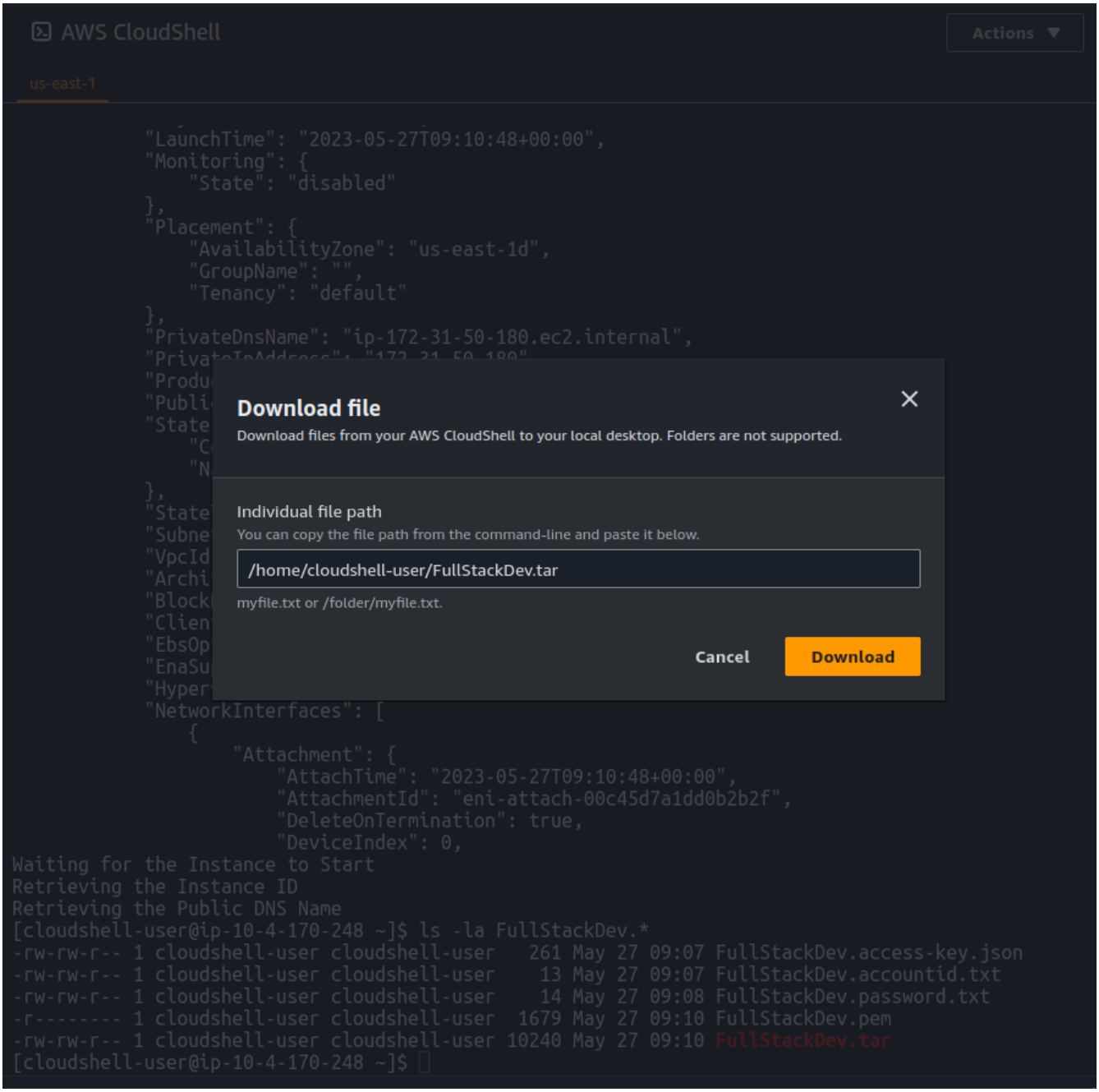
```
-rw-rw-r-- 1 cloudshell-user cloudshell-user   261 May 27 21:21 FullStackDev.access-key.json
-rw-rw-r-- 1 cloudshell-user cloudshell-user    13 May 27 21:21 FullStackDev.accountid.txt
-rw-rw-r-- 1 cloudshell-user cloudshell-user    49 May 27 21:25 FullStackDev.DevMachine1
-rw-rw-r-- 1 cloudshell-user cloudshell-user    14 May 27 21:21 FullStackDev.password.txt
-r-------- 1 cloudshell-user cloudshell-user  1679 May 27 21:24 FullStackDev.pem
-rw-rw-r-- 1 cloudshell-user cloudshell-user 10240 May 27 21:25 FullStackDev.tar
```

We may want to save the file `FullStackDev.tar` if we want keep our virtual machine.

In order to download this file, click in `Actions/Download file`, as shown in the following figure:



Next, specify the name of the file to download:



Logging into the instance via SSH

```
{
  echo Entering the Instance via SSH
  ssh -i FullStackDev.pem ubuntu@$public_dns_name
}
```

If you saved the `FullStackDev.tar` file on a local machine with a Linux environment in a directory called `~/Downloads`, you could try the following steps to gain remote access to the `DevMachine1` machine we just created:

```
{
  cd ~/Downloads
  tar xf FullStackDev.tar
  ssh -i FullStackDev.pem $(<FullStackDev.$my_machine_name)
}
```

Checking the software versions

After following the last section, you should have your Linux machine created. It will be an environment mounted with a Ubuntu version 22.04.

So, the following are the only prerequisites that we need to attend before starting to code in Java:

1. An Bash terminal.
2. Zip and Unzip installed (to install SDKMAN).
3. Git installed and configured.
4. Java installed.

These are the expected versions of the software prerequisites, after the machine creation:

Command	Expected Ouput
`echo $BASH_VERSION`	`5.1.16(1)-release`
`git --version`	`git version 2.34.1`

Installing more packages

Install some other required packages:

```
{
  sudo apt update

  # required by SDKMAN:
  sudo apt -y install zip unzip

  # required by GraalVM native-image:
  sudo apt -y install gcc zlib1g-dev
}
```

In a Ubuntu 22.04 local machine environment (your actual machine, outside the AWS), you could do our next steps by creating a local user `dev` and setting a password for it:

```
{
  sudo useradd -m -s /bin/bash -G sudo dev
  sudo passwd dev
}
```

After that, you could use `sudo` to logging in with:

```
sudo su - dev
```

Configuring Git

```
{
  confirm() {
    local prompt=$1
    local varname=$2
    while :
    do
      read -p "$prompt: " myvar
      read -p "Confirm (Y/n)? " -n 1 -r
      echo
      [[ ${REPLY:-y} =~ ^[Yy]$ ]] && {
        eval $varname="'$myvar'"
        break
      }
    done
  }
  confirm user.name my_user &&
    git config --global user.name "$my_user"
  confirm user.email my_user &&
    git config --global user.email "$my_user"
  git config --global init.defaultBranch main
}
```

Verify your configurations:

```
git config --global --list
```

Installing SDKMAN

```
{
  curl -s "https://get.sdkman.io" | bash
  source ~/.sdkman/bin/sdkman-init.sh
}
```

Installing Java (GraalVM version)

```
sdk install java 22.3.r19-grl && java -version
```

Installing Node.js (with GraalVM)

Install the Node.js component, check the location of `node` and `npm` commands and their versions:

```
{
  gu install nodejs
  echo
  for cmd in node npm
  do
    echo -n "$cmd (location / version): "
    which $cmd | tr -d '\n'
    echo -n " / "; $cmd -v
  done
}
```

2.1.5. → Part 2 ←

Building the classic "Hello World!" in Java

We will create a simple Java code that prints a "Hello World!".

To do this using our opened terminal, **copy and paste** the following commands to it:

```
{
  mkdir -p simple-tutorial/java && cd $_
  cat <<'EOF' > HelloWorld.java
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World!");
    }
}
EOF
}
```

Compile and run the code:

```
javac HelloWorld.java && java HelloWorld
```

Install GraalVM `native-image`:

```
gu install native-image
```

Build ant test the `helloworld` binary:

```
native-image HelloWorld
```

This takes a minute to execute:

```
========================================================================================================================
GraalVM Native Image: Generating 'helloworld' (executable)...
========================================================================================================================
[1/7] Initializing...                                                                            (7.5s @ 0.20GB)
 Version info: 'GraalVM 22.3.0 Java 19 CE'
 Java version info: '19.0.1+10-jvmci-22.3-b08'
 C compiler: gcc (linux, x86_64, 11.3.0)
 Garbage collector: Serial GC
[2/7] Performing analysis...  [****]                                                            (31.0s @ 0.69GB)
   2,909 (73.55%) of  3,955 classes reachable
   3,453 (50.75%) of  6,804 fields reachable
  12,988 (43.29%) of 30,004 methods reachable
      25 classes,     0 fields, and   330 methods registered for reflection
      57 classes,    56 fields, and    52 methods registered for JNI access
       4 native libraries: dl, pthread, rt, z
[3/7] Building universe...                                                                       (3.9s @ 0.38GB)
[4/7] Parsing methods...      [**]                                                               (3.4s @ 1.02GB)
[5/7] Inlining methods...     [***]                                                              (2.5s @ 0.60GB)
[6/7] Compiling methods...    [*****]                                                           (26.6s @ 0.35GB)
[7/7] Creating image...                                                                          (2.2s @ 0.72GB)
   4.17MB (35.71%) for code area:     7,334 compilation units
   6.96MB (59.58%) for image heap:   93,758 objects and 5 resources
 563.51KB ( 4.71%) for other data
  11.68MB in total
------------------------------------------------------------------------------------------------------------------------
Top 10 packages in code area:                        Top 10 object types in image heap:
 646.56KB java.util                                   927.94KB java.lang.String
 327.02KB java.lang                                   922.83KB byte[] for code metadata
 269.40KB java.text                                   831.48KB byte[] for general heap data
 221.24KB java.util.regex                             639.05KB java.lang.Class
 192.93KB java.util.concurrent                        554.41KB byte[] for java.lang.String
 149.41KB java.math                                   227.27KB c.o.svm.core.hub.DynamicHubCompanion
 118.26KB java.util.stream                            217.70KB java.lang.Object[]
 115.54KB java.lang.invoke                            196.73KB java.util.HashMap$Node
 114.96KB com.oracle.svm.core.genscavenge             164.27KB java.lang.String[]
  92.94KB java.util.logging                           157.17KB j.util.concurrent.ConcurrentHashMap$Node
   1.92MB for 124 more packages                          1.58MB for 782 more object types
------------------------------------------------------------------------------------------------------------------------
              1.6s (1.9% of total time) in 22 GCs | Peak RSS: 1.65GB | CPU load: 1.98
------------------------------------------------------------------------------------------------------------------------
Produced artifacts:
 /home/ubuntu/simple-tutorial/java/helloworld (executable)
 /home/ubuntu/simple-tutorial/java/helloworld.build_artifacts.txt (txt)
========================================================================================================================
Finished generating 'helloworld' in 1m 19s.
```

Test the `helloworld`:

```
./helloworld
```

In terms of performance, a native image is much superior:

```
$ time java HelloWorld
Hello World!

real    0m0.076s
user    0m0.085s
sys     0m0.014s

$ time ./helloworld
Hello World!

real    0m0.003s
user    0m0.003s
sys     0m0.000
```

Check the `helloworld` binary dependencies:

```
ldd helloworld
```

This is what we expect to see as a result:

```
linux-vdso.so.1 (0x00007ffea9353000)
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f2438e40000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f2437e00000)
/lib64/ld-linux-x86-64.so.2 (0x00007f2438e77000)
```

Initialize a new Git repository and make your first commit:

```
{
  cd ..
  git init
  cat > .gitignore <<EOF
*.class
/java/helloworld*
EOF
  git add .
  git commit -m 'Made my first commit. Added HelloWorld.java'
}
```

Building the classic "Hello World!" in JavaScript

```
{
  mkdir javascript && cd $_
  cat > HelloWorld.js <<'EOF'
console.log('Hello World!')
EOF
  node HelloWorld.js
}
```

Let's do all last `git commit` in this tutorial:

```
git add -A && git commit -m 'Added HelloWorld.js'
```

2.1.6. → Part 3 ←

Deleting the AWS instance

First, exit your VM.

Type `Ctrl` + `D` or `exit`.

Next, in the `cloudshell` terminal, **copy and paste** the following commands:

```
{
  echo Retrieving the Instance ID
  instance_id=$(aws ec2 describe-instances --filters \
    "Name=tag:Name,Values=$my_machine_name" --query \
    'Reservations[].Instances[].InstanceId' --output text)

  [ "$my_machine_security_group" ] || {
    echo Retrieving the Security Group ID
    my_machine_security_group=$(aws ec2 describe-instances \
      --instance-ids $instance_id --query \
      'Reservations[].Instances[].SecurityGroups[].GroupId' --output text)
  }

  echo Retrieving the Key Pair Name
  key_pair_name=$(aws ec2 describe-instances \
    --instance-ids $instance_id --query \
    'Reservations[].Instances[].KeyName' --output text)

  echo Terminating the Instance
  aws ec2 terminate-instances --instance-ids $instance_id

  echo Waiting for the Instance to Terminate
  aws ec2 wait instance-terminated --instance-ids $instance_id

  echo Deleting the Tags associated with the Instance
  aws ec2 delete-tags --resources $instance_id

  echo Deleting the Security Group
  aws ec2 delete-security-group --group-id $my_machine_security_group

  echo Deleting the Key Pair
  aws ec2 delete-key-pair --key-name $key_pair_name
}
```

Deleting the AWS user

```
{
  unset AWS_ACCESS_KEY_ID AWS_SECRET_ACCESS_KEY

  echo Removing the Access Keys
  access_keys=$(aws iam list-access-keys --user-name FullStackDev \
    --query 'AccessKeyMetadata[].AccessKeyId' --output text)
  for key_id in $access_keys
  do
    aws iam delete-access-key --user-name FullStackDev \
      --access-key-id $key_id
  done

  echo Detaching Policies
  for policy in \
    AmazonEC2FullAccess AWSCloudShellFullAccess IAMReadOnlyAccess
  do
    aws iam detach-user-policy --user-name FullStackDev \
      --policy-arn arn:aws:iam::aws:policy/$policy
  done

  echo Deleting the login profile
  aws iam delete-login-profile --user-name FullStackDev

  echo Deleting the User
  aws iam delete-user --user-name FullStackDev
}
```

That's it for this tutorial.
Thanks for following it!

2.2. References

AWS:

1. **AWS CloudShell** → https://aws.amazon.com/cloudshell/
2. **sign-in-url-defined** → About sign-in URLs
   https://docs.aws.amazon.com/signin/latest/userguide/sign-in-urls-defined.html

GraalVM:

1. Node.js Runtime
   https://www.graalvm.org/22.0/reference-manual/js/NodeJS/
2. How to create a node.js Native Image with GraalVM
   https://stackoverflow.com/questions/53734585/how-to-create-a-node-js-native-image-with-graalvm
3. Build a Native Executable
   https://www.graalvm.org/22.2/reference-manual/native-image/

2.3. Thank you for buying this tutorial ☺! Now you got more!

You read the **paid version** of this tutorial, which offers you (my customer) some other benefits than the public version.

Here are the benefits that I promised to you:

1. **Content customized for you**: did you notice your name at some points in this text?
2. **Facility to copy and paste commands**: did you notice the **copy and paste** button in the HTML version? Also, did you see that each block code in the PDF version (presented as an image) is clickable? When you click the figure of the source code, a window browser is opened, and then you can copy it.
3. **Access to the Git repository**: after you pass me your GitHub user account, you will have access to the private Git repository of tutorials like this.
4. **No advertisement**: did you see any ad in this version? ☺
5. **Extra content**: did you notice the References section? Also, updates to this tutorial will receive more private content soon.
6. **A PDF version**: You are reading it! ☺
7. **Web3 Tokens**: You will get more information about how to gain these Web3 tokens in the following tutorials.
8. **Direct support**: Please, subscribe to my private Discord channel to get my support.

See the public version

3. Want to know more about how to write texts (tutorials, ebooks) like this one?

3.1. I can indicate some good tools to you

Here are some of the tools that I have experience in use to create my texts:

Category	Tools
Text formats	1. Markdown and GFM 2. MDX 3. AsciiDoc
Text processors	1. Txt2tags 2. Asciidoctor (Ruby version) including: a. Asciidoctor.js (JavaScript version) including many extensions like my extension b. AsciidoctorJ (Java version)
Text Editor	1. Neovim
Artificial Intelligence	1. Google Translate 2. DeepL Translate 3. Grammarly
Programming languages	1. Bash 2. Python 3. JavaScript 4. Java 5. Groovy 6. Ruby

3.1.1. My gen-code extension

1. Is a Asciidoctor.js Preprocessor Extension.
2. The source of this document (sample.adoc) uses it which is responsible for:
   a. Generate source files from source code that are inside some source blocks.
   b. Generate attributes used by the document itself.
   c. Convert these generated codes to images using tools like Silicon and Text2png.
   d. Generate another document sample.adoc (public) that only refer to the images created.
3. Currently, its source code is unavailable, but it will soon be open source.

3.2. I can offer you a consultancy ...

... in which I will give you my tips about the tools I use to create my documents and/ or we'll create some documents together on a topic of your need.

3.3. You can buy `my docs-as-code course` (it will be available soon)

In my course, we will explore how the AsciiDoc format is used in some of my `docs-as-code` projects, including these:

1. The project that builds this document you are reading.
    a. It is created with some code in Bash and JavaScript (running on Node.js).
    b. It uses my extension.
    c. Currently, the generated PDF version is created by asciidoctor-pdf (which uses a Ruby execution environment). But, until the availability of this course, I intend to migrate it to use Asciidoctor Web PDF.
2. The project of my WebSite.
3. The project of my CV.
4. Some AsciiDoc slides, like these:
    a. https://paulojeronimo.com/asciidoc-dzslides-slides/

So, based on the code of the projects above, we will see how the following tools are used to build AsciiDoc documents:

1. Neovim ← My favorite text editor.
2. Bash ← We'll use how I use it to build these docs.
    a. docker-asciidoctor-builder ← A simple Bash tool that I developed and use a lot
3. Asciidoctor (Ruby version).
4. Asciidoctor.js and some extensions (including my extension).
5. GitHub Pages.

As a bonus, I also intend to show you a little about the following tools:

1. Asciidoctor Web PDF.
2. Antora.