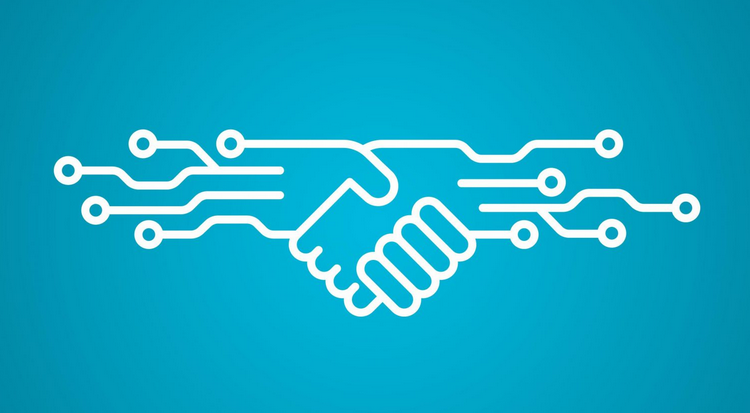
**Plutus: Learning a**

**smart contract language**



**IOG**

**Plutus: Learning a smart contract language**

Copyright © March 2023, IOG (Input Output Global)

This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc/4.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

This book was prepared by IOGs education team. Majority of the text is taken from the Plutus pioneer program 4rd iteration that was presented by Lars Brünjes and his colleagues. All program code in this book is taken from the Plutus pioneer program and is freely available at:

<https://github.com/input-output-hk/plutus-pioneer-program/tree/fourth-iteration>

All pictures in this book are the copyright of IOG if not otherwise noted. The picture on the front page is owned by cornellilj.org and published on following blog post:

<https://cornellilj.org/2018/02/08/smart-contracts-another-feather-in-uncitrals-cap/>

Table of Contents

[1 Plutus introduction 5](#_Toc128476795)

[1.1 Setting up your development environment 5](#_Toc128476796)

[1.1.1 Using Demeter.run 5](#_Toc128476797)

[1.1.2 Using Docker 5](#_Toc128476798)

[1.2 Kuber marketplace 6](#_Toc128476799)

[1.3 Hashing and digital signatures 6](#_Toc128476800)

[1.4 The EUTxO model 6](#_Toc128476801)

[1.5 Plutus code 8](#_Toc128476802)

[2 Resources 10](#_Toc128476803)

Table of Figures

Figure 1 - UTXO model [1] 6

# Plutus introduction

Plutus is the native smart contract language for Cardano. It is a Turing-complete language written mostly in Haskell, and Plutus smart contracts are effectively Haskell programs. By using Plutus, you can be confident in the correct execution of your smart contracts. It draws from modern language research to provide a safe, full-stack programming environment based on Haskell, the leading purely functional programming language. [1]

So, in order to understand the code in this book and one must understand the basics of the Haskell programming language. The Haskell project contains a list of learning resources as books and tutorials under its Documentation section, from which some are free and some commercial. It can be found at Haskell’s official web page:

* <https://www.haskell.org/documentation/>

|  |  |
| --- | --- |
|  | IOG provides its own Haskell course that is tailored to the needs of learning Marlowe, also a smart contract langauage and Plutus. It can be found at IOGs GitHub page:  <https://github.com/input-output-hk/haskell-course> |

If you would like to learn more about Plutus and its software development kit (SDK) in addition to what this book offers, you can check out the online documentation which can be found at:

* <https://plutus.readthedocs.io/en/latest/>
* <https://plutus-apps.readthedocs.io/en/latest/>

## Setting up your development environment

The installation instructions for setting your development environment are split in two parts. The first part is using the online platform *Demeter.run* and the second part is setting up a Docker container locally on your PC. The second instructions are general and can be used on Windows, Mac OS and Linux operating systems.

### Using Demeter.run

*Demeter.run* is a cloud-based environment that provides all the tools required for building and deploying Cardano applications. It can be found at <https://demeter.run/>. To use demeter.run, you need a web browser on any operating system and internet access. Follow the steps below to set up your demeter.run account and to use this development environment.

To setup a demeter.run account and start using it, follow the instructions below:

1. Open your browser and navigate to https://demeter.run/. You'll see the home page of demeter.run as the image below shows. There click on the "Login" option on the top navigation menu, as seen in the image below.

Graphical user interface, application, Teams

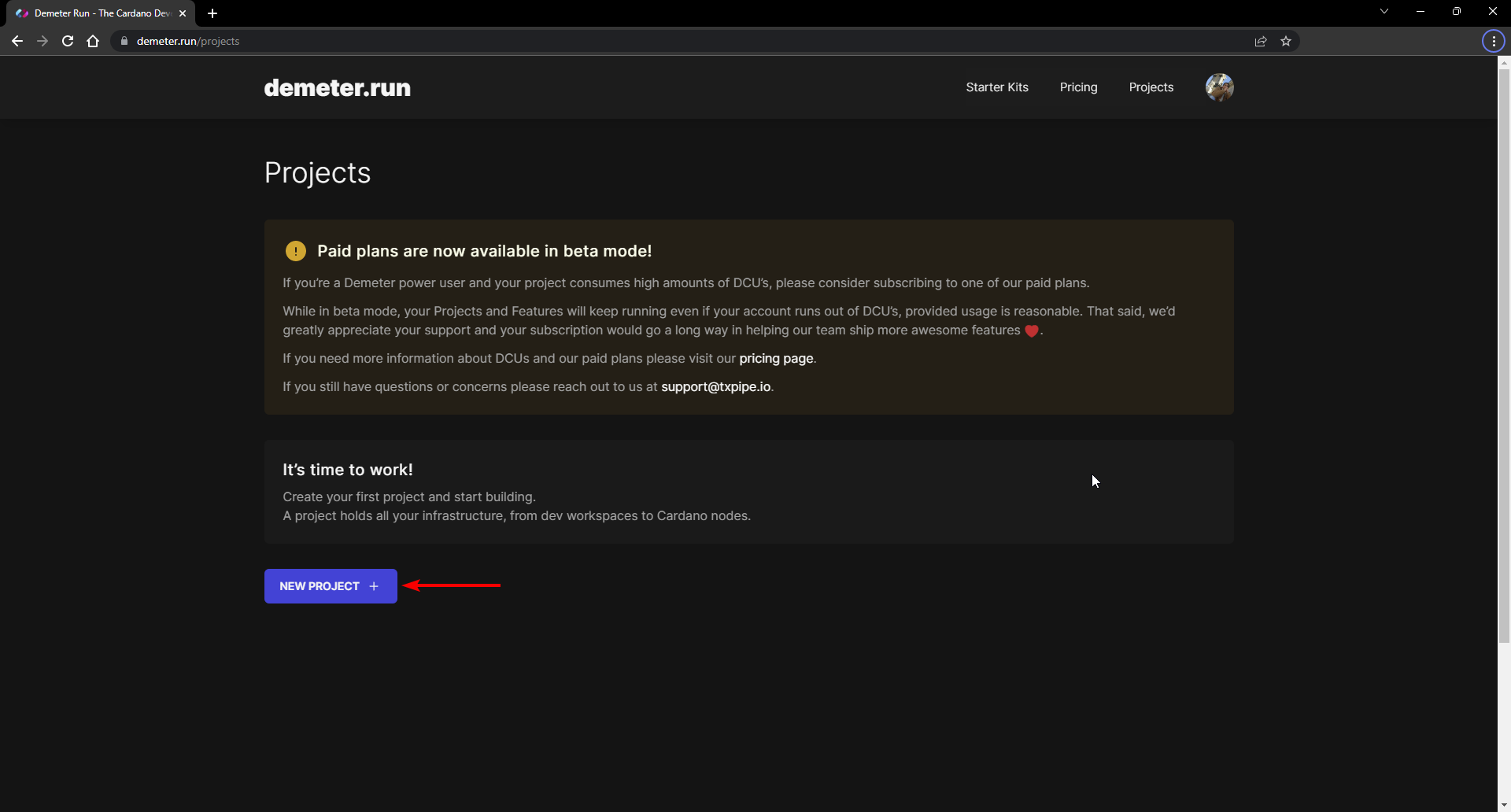
Description automatically generated

1. Next, you need to set up an account. As it's shown in the image below, you can choose to set a username and password, or you can sign in by using your Google or GitHub account. Select the method that best fits your preferences to continue.

A screenshot of a computer

Description automatically generated

1. After signing in, you're ready to use demeter.run. As the image below shows, you'll see the "Projects" page. There click on the "NEW PROJECT" button.



1. Next, you'll need to follow the required steps to set up a new project. First, you need to choose an organization where your project will reside. By default, an organization with your username exists. We'll select the default organization for this demo, as shown in the image below, where the username is “jarturomora”. After choosing the organization, click on the "NEXT" button.

A screenshot of a computer

Description automatically generated with medium confidence

1. In this step you need to choose the location of the cluster that you'll use. As for now, only a US-based cluster exists. So, select the "US Central" cluster and click the "NEXT" button to continue, as the image below shows.

A screenshot of a computer

Description automatically generated with medium confidence

1. After choosing a cluster, the next step is to select a plan. For the purpose of the program, we will use the "Discover" plan that is available free of charge. So, select the "Discover" plan and click on the "NEXT" button to continue, as the image below shows.

A screenshot of a computer

Description automatically generated with medium confidence

1. Next, you need to choose a network for your project. For testing purposes, we'll typically use the "Preview" networks. Select the "Preview" network and click the "NEXT" button to continue, as the image below shows.

A screenshot of a computer

Description automatically generated with medium confidence

1. The last step is to provide a name for your project. Also, in this step, you can review the project's details; be aware that you can have only one project in the Discover plan. In this demo, the project was named "PPP Demo." So, set a name for your project and click on the "CREATE PROJECT" button to finish, as the image below shows.

A screenshot of a computer

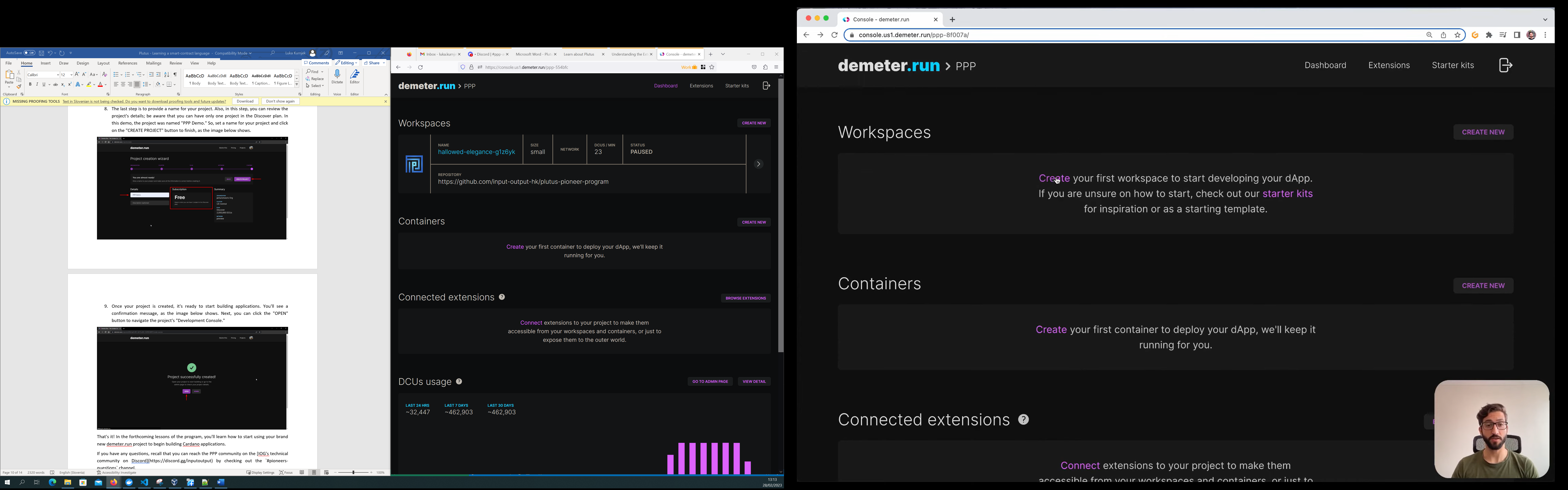
Description automatically generated with medium confidence

1. Once your project is created, it's ready to start building applications. You'll see a confirmation message, as the image below shows. Next, you can click the "OPEN" button to navigate the project's "Development Console."

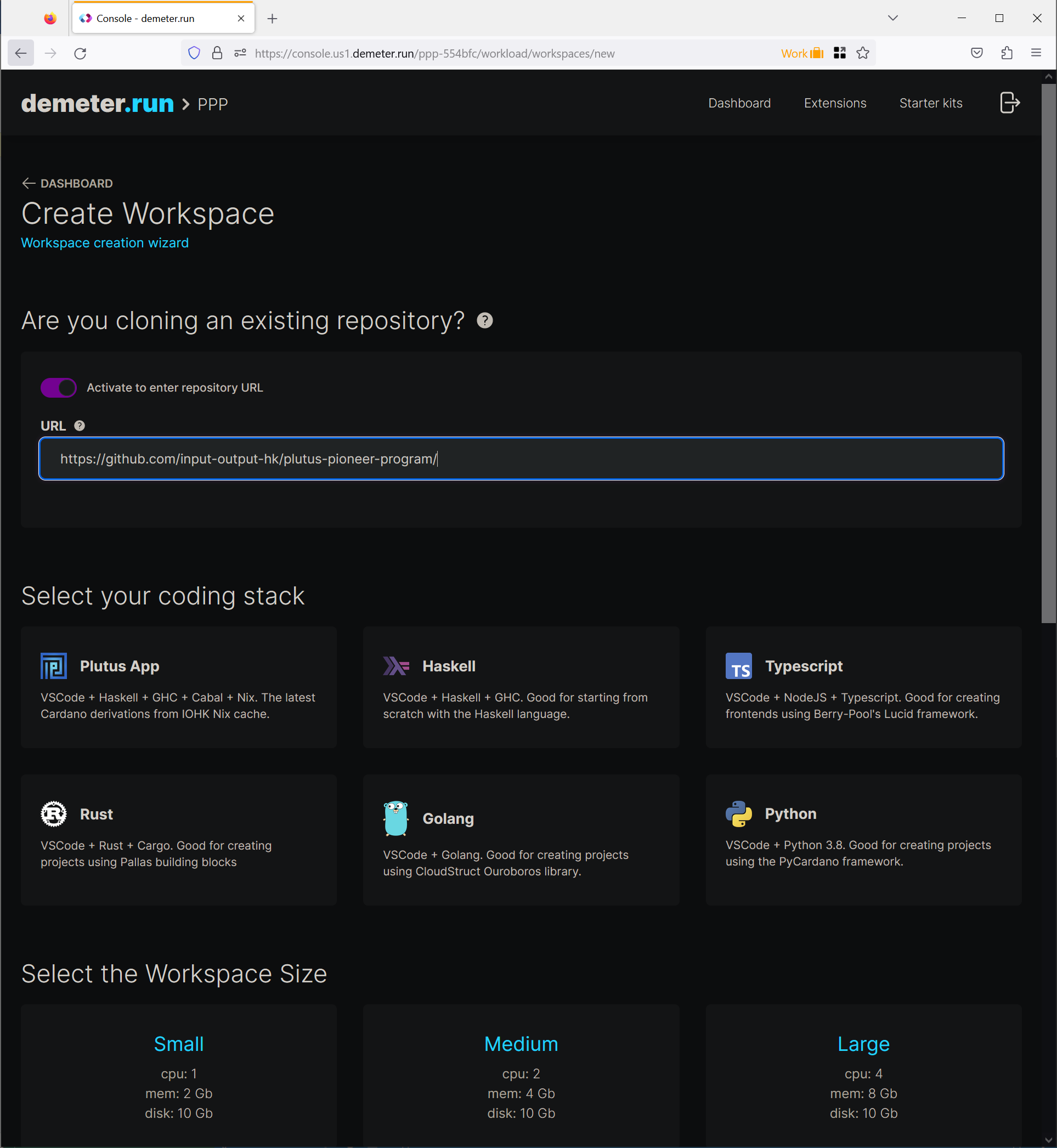
A screenshot of a computer

Description automatically generated

1. After that you see the dashboard. You will need to create now a workspace that is tailored to your needs of development tools. Click on Create new on the upper right corner of the Workspace section.



1. In the Create workspace section we click on the toggle button for using an existing repository and input the Plutus Pioneer Program GitHub repository. If you have a GitHub account, you can clone the PPP repo to your account and the input a link to your repo. In the “Select coding stack” section choose the Plutus App option.



1. At the bottom of this page, you can select your workspace size. The bigger the size is the more credits it consumes. For our project we will choose medium size. You have 2.000.000 DCU (Demeter Computed Units) for free. For the network we select Preview and then click on the Create button at the end of the page.

Graphical user interface, website

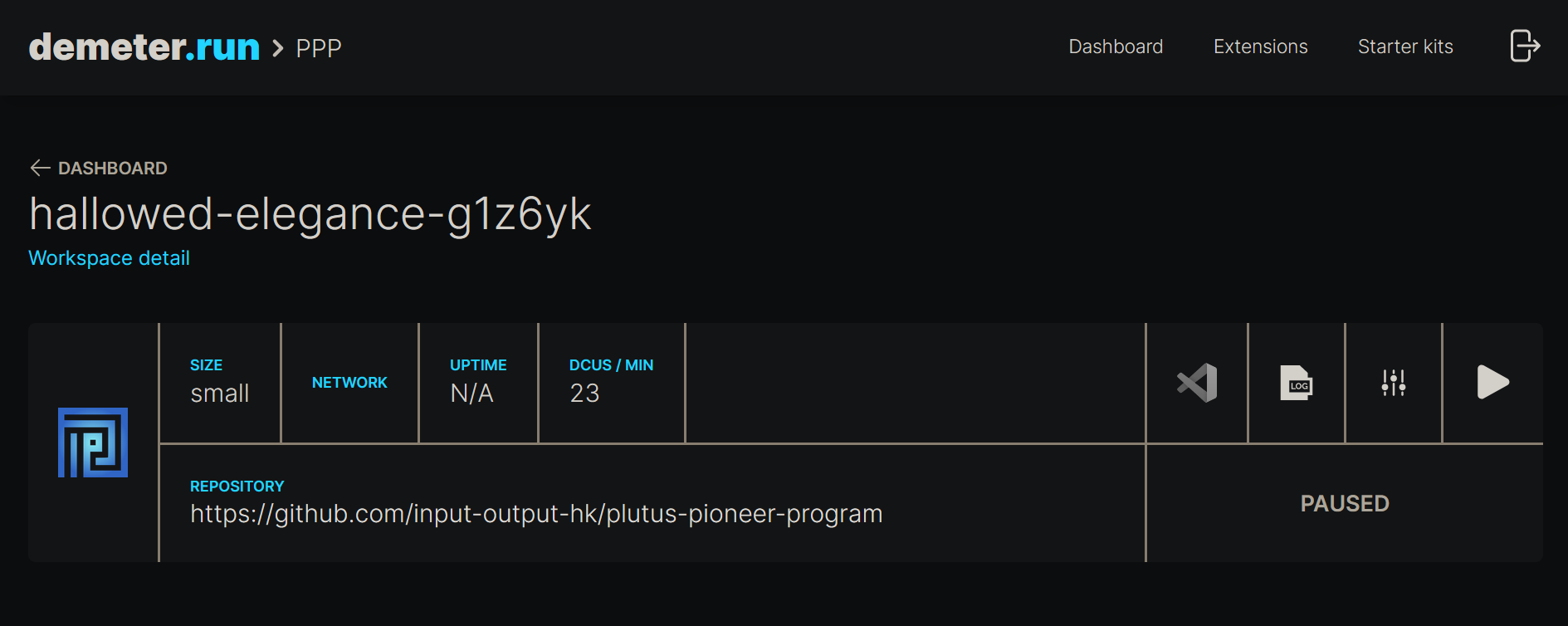
Description automatically generated

A screenshot of a computer screen

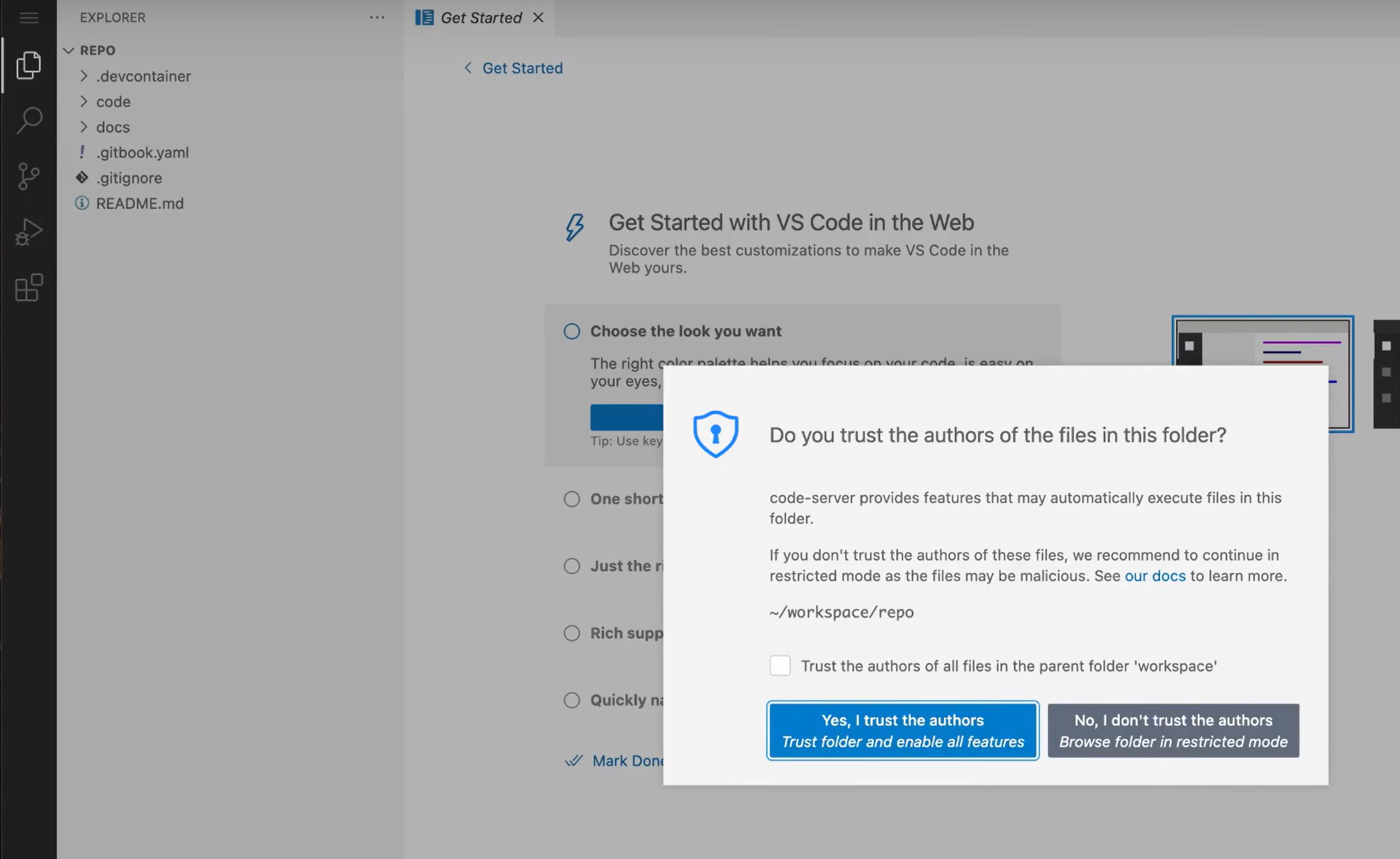
Description automatically generated with medium confidence

1. After that you will be redirected to the workspace detail page and the workspace information will be displayed. The workspace also needs some time before it builds. After its build, it will automatically go into run mode. You can click on the “Pause workspace” button on the right side to pause it and then click again on the same button that will have the tool tip ““Run workspace” to run it again.

|  |  |
| --- | --- |
|  | Demeter.run will charge you DCUs for a running workspace. So after you finish with your work always pause your workspace. |



1. Once everything is setup you can click on the VSCode logo and a VSCode editor will appear that will contain all the files from the pioneer program repository you inputted in the step when creating your workspace. When it will ask you if you trust the authors of this project check the trust box and click on the Yes button.



1. Once you are inside the VSCode editor you can click on the menu button in the upper left corner and click on Terminal -> New Terminal. This will open a terminal window in your workspace and certain command line tools will be available. The code for a lecture is contained in the *code/WeekXX/lecture* folders.

A screenshot of a computer screen

Description automatically generated

1. With the following commands you can update your project and create a validator script for the *FortyTwo.hs* code example from the Week02 folder.

$ cd code

$ cabal update

$ cd Week02

$ cabal repl

Prelude Gift> import FortyTwo

Prelude FortyTwo Gift> FortyTwo.saveVal

The file *fortytwo.plutus* is saved in the assets folder. You will learn in the next chapter what you can do with the code in the Week02 folder. If you have any questions, you can reach the PPP community on the IOG's technical community on Discord (<https://discord.gg/inputoutput>) by checking out the #pioneers-questions channel.

### Installing Docker and VSCode

In this guide, you'll learn how to set up a local working environment using Docker and Visual Studio Code. If you have no previous experience using Docker and VSCode, you can follow the steps in this chapter that explain all necessary actions for a successful environment setup.

|  |  |
| --- | --- |
|  | Use a host computer, not a virtual machine. The setup instructions will only work if you use Docker Desktop or Engine on your host computer. The setup inside a virtual machine will fail, whether it's a Windows or a Linux virtual machine. |

Docker is a platform designed to help developers build, share, and run applications in an isolated environment on any operating system. To ease setting up a local working environment for this course, the IOG Education Team created a Docker container that packages all the dependencies required to follow up the lessons of this program.

A Docker container is a standard unit of software that packages up code and all its dependencies, so an application runs quickly and reliably from one computing environment to another. From the Docker documentation, you can learn more about Docker containers on the following page: <https://www.docker.com/resources/what-container/>.

Follow the next steps to install Docker in your computer:

1. Open your browser and navigate to <https://www.docker.com>. Click the "Download Docker Desktop" button from the Docker's homepage. By default, you'll download a version compatible with your operating system. Docker is available for Linux, Microsoft Windows, and Apple macOS.

|  |  |
| --- | --- |
| Icon  Description automatically generated | Important note for macOS users. Be sure that you download the correct version according to the chip of your computer, for M1 or M2 chips, download the "Apple Chip" version. For Intel chips, download the "Intel Chip" version. |

1. After downloading the Docker Desktop installer, execute it and follow the instructions by choosing the default options. Installation options may vary depending on your chip and operating system. If you need detailed instructions, please visit the Get Docker section on the docker docs website: <https://docs.docker.com/get-docker/>.
2. When Docker Desktop is started, it automatically starts the docker daemon in the background. This will happen every time you login into your computer. You can change this behavior by turning off the "Start Docker Desktop when you log in" in the Docker Settings configuration. Note that the docker daemon must be running when accessing the docker container in VSCode which we will explain next.

Graphical user interface, text, application

Description automatically generated

Visual Studio Code (<https://code.visualstudio.com/>), also known as VS Code, is a source code editor freely distributed by Microsoft that runs on Windows, Linux, and macOS. Additionally, to code editing, VS Code allows you to create and install extensions that eases your daily work.

Don't install any Haskell extension in Visual Studio Code. If you have VS Code installed, you may see several prompts from VS Code asking for permission to install several Haskell extensions when you open the PPP repository. You don't need to install any of those, as the Docker container we deliver will install all the Haskell extensions that VS Code needs. Some Pioneers who installed additional Haskell extensions report issues in completing this install guide; also, IOG's Education Team members experienced problems while compiling Plutus scripts.

Follow the next steps to install VS Code and a handy extension that you will use in this course:

1. Open your browser and navigate to <https://code.visualstudio.com/> to open the Visual Studio Code website. There is a download button that suggest you to download the software depending on your operating system. Be sure to download the latest version.
2. After downloading the VS Code installer, execute it and follow the instructions by choosing the default options. Installation options may vary depending on your chip and operating system. If you need detailed instructions, please visit the Setting up Visual Studio Code section in the VS Code docs website:

<https://code.visualstudio.com/docs/setup/setup-overview>

1. Once you have installed VS Code, open it to install an extension. Extensions are additional add-ons that extend the VS Code's functionality; Microsoft provides some extensions, but also, plenty of extensions are created by other companies and developers. To install an extension, click on the "Manage" icon in the bottom left corner and choose the "Extensions" options in the image below shows.

A screenshot of a computer

Description automatically generated with medium confidence

1. Next, a window showing the "Extensions Marketplace" will appear on the left side of the VS Code UI. In the search box, type “remote development” to look for the Remote Development extension provided by Microsoft:

<https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack>

Once the extension appears, click on it. As the image below shows, you should click on the "Install" button to install an extension.

A screenshot of a computer

Description automatically generated with medium confidence

After successfully installing the Remove Development extension, you'll note that an "Uninstall" button appears in the extension description tab, and the "Open a Remote Window" green icon will appear on the bottom left corner of the VS Code UI as you can see in the following image.

You have now installed the required software to use the Docker container. Next, we'll guide you through the steps you must follow to load the Docker container and execute it for the first time.

### Running the PPP Docker Container

Before moving forward into using the Docker container provided for this cohort of the PPP, you need to close VS Code. Next, please follow the next section, where we'll guide you on finishing the setup of your local working environment.

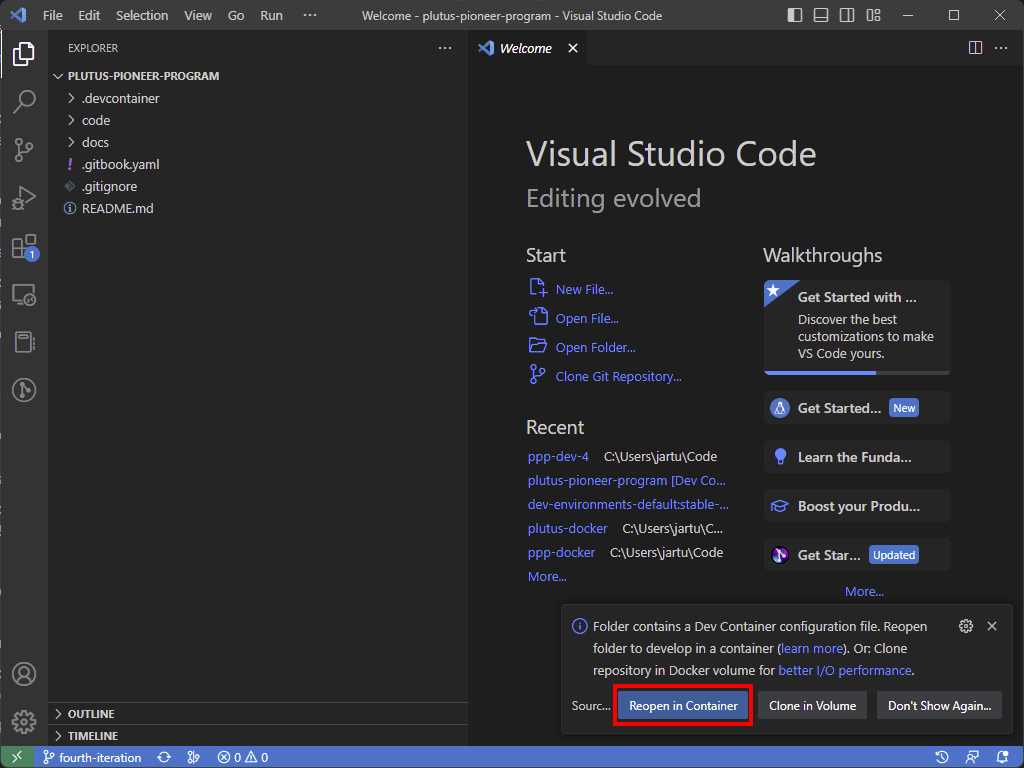
As stated in the demeter.run section you can work with the official GitHub repository or create a fork if you have a GitHub account and want to keep your work online. Instructions on how to create a GitHub fork can be found in the Plutus Pioneer GitBook at the following location:

<https://iog-academy.gitbook.io/plutus-pioneers-program-fourth-cohort/preliminary-work/setup/docker#forking-the-ppp-repository>

In either case you will need to clone a Plutus pioneer repository to your computer using Git. If you don't have Git installed, please follow the instructions from the Git documentation <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>, where you'll find detailed information about installing Git on Linux, macOS or Windows. Once you have Git installed open up a terminal that has the git command available, cd into you desired location and clone the PPP GitHub repository:

$ git clone https://github.com/input-output-hk/plutus-pioneer-program

Next open VSCode and in the File menu click on Open Folder and then select the PPP folder you have cloned with git. When VS Code opens, the "Remote Development" extension will detect the Docker container. As shown in the image below, you'll see a message asking to reopen your project in the container. Click on the "Reopen in Container" button to continue.



After reopening your project in Container, the Docker container will be built. You can click on the "Starting Dev Container" message to view the log. Please look at the log to be aware of when the container is ready. The build may take some time. After a few minutes, you will see a message in the log starting with the text Start: Run in container as you can see in the image below. This message indicates that the dev container is ready.

Text

Description automatically generated

You can safely close this terminal window by clicking on the trash can icon in the upper right corner of the terminal window. Now, open a new terminal window into VS Code by clicking on the "Terminal" menu and then “New Terminal”. From the new terminal window cd into the *code* and update the repository with the following command:

root@99286bb23f1c:/workspaces/plutus-pioneer-program# cd code

root@99286bb23f1c:/workspaces/plutus-pioneer-program/code# cabal update

This step is critical as all the code and updates should run into the code directory. **Be patient while running these commands.** Depending on your hardware configuration and an internet connection, the time required to execute the command cabal update may vary. It takes at least 5 minutes to finish; however, we experienced waiting times of up to 15 minutes in some hardware and internet settings. Now, to finish the dev container setup, type and execute the following command in the VS Code terminal to build all the dependencies required by Plutus.

root@99286bb23f1c:/workspaces/plutus-pioneer-program/code# cabal build all

After successfully running this command, you will see the system prompt back with no errors. Also this command can take at least 10 minutes to finish. However, we experienced waiting times of up to 25 in some hardware and internet settings.

## Kuber marketplace

## Hashing and digital signatures

## The EUTxO model

The Cardano blockchain uses the Extended UTXO model that is a variant of the Unspent Transaction Output (UTXO) model used by Bitcoin. Transactions consume unspent outputs (UTXOs) from previous transactions and produce new outputs, which can be used as inputs to later transactions. Unspent outputs are the liquid funds on the blockchain. Users do not have individual accounts, but rather have a software wallet on a smartphone or PC which manages UTXOs on the blockchain. It can initiate transactions involving UTXOs owned by the user. Every core node on the blockchain maintains a record of all the currently unspent outputs, the UTXO set. When outputs are spent, they are removed from the UTXO set.

Diagram

Description automatically generated

Figure 1 - UTXO model [1]

There are other models than UTXO. Ethereum, for example, uses a so-called account-based model, which is what a normal bank uses. There everybody has an account, and each account has a balance. If you transfer money from one account to another, then the balances get updated accordingly. But in the UTXO model, the input is always the entire balance of an UTXO, and the outputs are newly created UTXOs from which one of them could be belonging to the user that provided his UTXO as input and would represent his change amount. For every transaction there is a fee to pay denominated in ADA for the Cardano blockchain.

As soon as an output is used as input in a transaction, it becomes spent and can never be used again. The UTXO output is associated to an address which is represented by a public key hash. We call them public key addresses. The ADA amount and optionally native tokens of an public key address is the sum of ADA and native tokens from all UTXOs belonging to this address. A transaction must be signed by the owner of the private key corresponding to the address that defines the input UTXO. Think of an address as a ‘lock’ that can only be ‘unlocked’ by the right ‘key’ ‒ the correct signature. The user which controls a private key of an address is able to create transactions and use the ADA or native tokens sitting at the UTXOs of this address.

The extended UTXO model introduces in addition to public key addresses also script addresses that can contain some logic. That logic defines under which conditions the UTXOs sitting at this address can be spent. The address is unlocked by, and piece of data called the *redeemer*, which in the conventional UTXO model would be a private key. A UTXO also contains some data called the *datum*, beside the amount of ADA sitting at the address. The datum together with the redeemer and the transaction context are the input information for a script logic that then chooses to weather this transaction is valid and can be processed by a node on the network.

You can check the validity of a transaction in your wallet. If it is valid, you can be sure it will be processed on the network, given the condition that all the UTXO inputs are still present at processing time. If they are not the transaction will simply fail and no fee will be charged to the user that sent the transaction. We call the script that validates a transaction the validator.

A transaction can be classified as a producing transaction that produces UTXOs or as a spending transaction that spends UTXOs. The truth is every transaction except the genesis transaction takes at least one UTXO as input and produces at least one UTXO as output. What we mean by the terms spending and producing is the context. Later in chapter 2 we will see code examples where we first send ADA to a script address and for this reason, we call it a producing transaction. And after that we try to collect that ADA from the script address, so we call this transaction a spending transaction.

The script address is defined as a hash of the validator code written in Plutus core language, which also needs to be provided. The script addresses are publicly known. The producing transaction must include this address, and it must include the datum or the hash of the datum that will be attached to the UTXO created at script address. If it includes the hash of the datum, only a person that knows the datum by some other means not by looking at the blockchain is able to ever spend such an UTXO. The spending transaction is responsible for providing the datum, the redeemer and the transaction context. It also provides the validator script. If we construct a transaction where the funds go to a public key address a datum is not needed.

As said the validator script takes the datum, the redeemer and the transaction context as input information. The input for the datum is collected from each UTXO individually that are sitting at a script address. That means if there are multiple UTXOs specified to be consumed in the transaction, the validation logic is checked for each of them separately. So, in each validation there is only one datum as input coming from one UTXO.

This limited view of the validator script that can see only inputs, outputs and the transaction that will be processed, has a security advantage compared to the Ethereum model, where the script can see the whole state of the blockchain. That enables Ethereum's scripts to be much more powerful but for this reason it's also very difficult to predict what a given script will do. That opens the door to all sorts of security issues. It can be mathematically proven that every logic you can express in Ethereum you can also express in the extended UTXO model. And that makes it a much safer and reliable transaction model compared to Ethereum.

## Plutus code

The code for Plutus smart contracts can be separated into two parts. First is the “on-chain” code, which consists of the validator function and some additional declarations and variables as the script address. This code gets compiled to Plutus Core language. It runs on the Cardano blockchain and once submitted it cannot be changed.

From the official documentation [1] we get the following description for Plutus Core:

Plutus Core is the scripting language used by Cardano to implement the EUTXO model. It is a simple, functional language similar to Haskell, and a large subset of Haskell can be used to write Plutus Core scripts. As a smart contract author, you don’t write any Plutus Core; rather, all Plutus Core scripts are generated by a Haskell compiler plugin called Plutus Tx.

The “off-chain” code can also be written in Haskell, just like the on-chain code. Unlike Ethereum where the on-chain code is written in Solidity, but the off-chain code is written in JavaScript. That way, the business logic only needs to be written once. This logic can then be used in the validator script and in the code that builds the transactions that run the validator script.

The off-chain code basically constructs the transaction and submits it to the blockchain. Since both the on-chain and off-chain code are written in Haskell they can reside in one Haskell file while testing your code, which allows them to share code between them. Another option is also to construct the off-chain transaction with command line tools instead of compiling Haskell code. Cardano provides for this the *cardano-cli* command line tool. There are also community alternatives. Those tools will also be presented in later chapters of this book. In the 4th iteration of the Plutus pioneer program we will not write any off-chain code but use only the command line tools. If you want to see examples of off-chain code written in Haskell you can look at the videos of the 3rd PPP iteration or also read the accompanied book. However, some of the code examples may not work today since the Plutus libraries were upgraded in the meantime.

# Resources

[1] The Cardano documentation

<https://docs.cardano.org/plutus/learn-about-plutus>

<https://web.archive.org/web/20220704234049/https://docs.cardano.org/plutus/eutxo-explainer>