Installing Prerequisites

In this section, we will set up our machine and install the prerequisites for developing, deploying and interacting with smart contracts on Cosmos SDK chains.

Go

You can set up Go following theofficial documentation . The latest versions ofwasmd require go versionv1.18+ .

Rust

Assuming you have never worked with Rust, you will first need to install some tooling. The standard approach is to userustup to maintain dependencies and handle updating multiple versions ofcargo andrustc , which you will be using.

Installing Rust in Linux and Mac

First,install rustup . Once installed, make sure you have the wasm32 target:

rustup default stable cargo version

If this is lower than 1.55.0+, update

rustup update stable

rustup target list --installed rustup target add wasm32-unknown-unknown info wasm32-unknown-unknown states the target specification, and is called "target triple" consisting of three strings separated by hyphens. Strings representarchitecture ,vendor , andoperating system respectively.unknown means there is no specifications for the target and the application is more flexible.

Read therust docs for a more detailed explanation.

Installing Rust in Windows 10

First, download and executerustup-init.exe fromrustup.rs orrust-lang.org .

If requested, manually download and install Visual C++ Build Tools 2019, fromhttps://visualstudio.microsoft.com/visual-cpp-build-tools/ . Make sure "Windows 10 SDK" and "English language pack" are selected.

Continue runningrustup-init.exe , and proceed with the installation.

Optionally:

- Download and installgvim
- , and modify the environment variables to add to
- the PATH.
- Download and installgit for windows
- . Modify the environment variables to add \bin to PATH.
- Turn on Developer Mode (Settings -> Update and Security: For Developers) and enable Device Discovery, to be able to access the Windows 10 server through ssh (https://www.ctrl.blog/entry/how-to-win10-ssh-service.html#section-mssshserv-enable
-).

Install the wasm32 target:

rustup default stable cargo version

If this is lower than 1.55.0, update

rustup update stable

rustup target list --installed rustup target add wasm32-unknown-unknown

wasmd

wasmd is the backbone of the CosmWasm platform. It is the implementation of a Cosmos zone with wasm smart contracts

enabled.

The code was forked from thecosmos/gaia repository as a basis, then x/wasm was added and many gaia-specific files were cleaned up. However, the wasmd binary should function just like gaiad except for the addition of the x/wasm module. As such,wasmd have all the same features (plus WASM smart contracts obviously). If you'd like to learn more about accessing those features take a look at theGaia docs . If you'd like to learn more about getting started with the Cosmos SDK in general, take a look at the series ofTutorials that show how to build custom modules for application-specific blockchains.

Run the following commands to installwasmd :

git clone https://github.com/CosmWasm/wasmd.git cd wasmd

If you are updating wasmd, first update your local repository by fetching the remote tags available

git fetch --tags

replace the v0.27.0 with the most stable version on https://github.com/CosmWasm/wasmd/tags (or look at git tag)

git checkout v0.27.0 make

install

verify the installation

wasmd version info * If you are on a Windows machine andmake * software is not preinstalled in the OS, install it following the instructionshere * . * make install * will copywasmd * toHOME/go/bin * by default. Still, if you have any problems with the installation ofwasmd * , check yourPATH * and make sure it includesHOME/go/bin * .

Setting up command-line tools

We will be using a few command-line tools extensively:

apt

install jq curl

Setting up your IDE

We will need a good IDE for developing smart contracts with Rust. The following development environments are highly recommended and coupling them with the corresponding Rust plugins will help you learn the syntax, especially if you have no prior experience.

- VSCode
- is the best supported environment for RLS (Rust Language Server) and coupled with the Use thust-analyzer
- extension instead (The previous plugin,RLS for VSCode
- , is deprecated.), it makes use of the rust compiler to type-check all your code on each save. This gives the same error messages as the actual compiler would and highlights the errors, but it can be a bit slow to respond (as it runs the compiler). It is a solid option, particularly if you are used to VSCode.
- The other option isIntelliJ IDEA Community Edition
- along with theIntelliJ Rust
- plugin, which has fast support for inline language features. In particular, it shows the inferred types of variables, which can be very helpful, especially when working with (nested) generics. It catches most syntax errors very quickly, but not all of them. This means sometimes you have to look at compile failures to find the errors. It is highly recommended, more so, if you are coming from another IntelliJ product (eg. Goland).

There are other editors out there with varying degrees of Rust support. However, unless you have a strong preference for another editor (e.g., Sublime, Emacs, Vim) trying one of the two above is recommended, especially if you are new to Rust. Once you are confident in the language, you can always use another editor and customize it to your liking.

CosmWasm IDE

CosmWasm IDE is a tool that simplifies CosmWasm smart contract development & deployment processes. It integrates with Gitpod & Keplr to create a simple yet powerful environment to build, deploy, and interact with CosmWasm smart contracts through default and custom networks using CosmWasm. With Gitpod, CosmWasm developers can develop smart contracts on the browser, and it is well maintained. The tool is currently maintained by Oraichain & CosmWasm.

Components

The CosmWasm IDE consists of three sub-repositories:

- CosmWasm Gitpod
- serves as a Gitpod builder which automatically builds a complete development environment including Rust installation, VS Code browser, crucial VS Code extensions, and is fully compatible with the Keplr wallet. With this repository, CosmWasm developers will not have to worry about spending hours installing tools & libraries, and they also feel secure when deploying contracts using Keplr.
- CosmWasm IDE extension
- is a VS Code extension which integrates all the important functionalities related to building & deploying CosmWasm smart contracts through simple button clicks.
- CosmWasm IDE extension webview
- is a React application that lies on top of the CosmWasm IDE Extension. It is responsible for connecting with the Keplr wallet and displaying inputs to deploy & interact with smart contracts. It also allows adding custom networks.

For more information, please visit the repositories on GitHub. You can also find a tutorial for CosmWasm IDE here or through the official Oraichain documentation site . Previous Your First Contract Next Setting up Environment * Go * Rust * * Installing Rust in Linux and Mac * * Installing Rust in Windows 10 * wasmd * Setting up command-line tools * Setting up your IDE