[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)/[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)



Search

[Interchain Developer Academy](https://ida.interchain.io/)[Interchain Developer Academy](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

Search



Filters

Interchain Developer Academy

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 0 - Getting Started](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Getting Started](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Blockchain 101](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Blockchain History](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Public and Managed Blockchains](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Consensus in Distributed Networks](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Cryptography](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Self-Assessment Quiz](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Go Introduction - First Steps](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Go Basics](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Go Interfaces](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Control Structures in Go](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Arrays and Slices in Go](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Standard Packages in Go](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Concurrency in Go](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Good-To-Know Dev Terms](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Docker Introduction](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Blockchain Technology and the Interchain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[The Interchain Ecosystem](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Getting ATOM and Staking It](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[A Blockchain App Architecture](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Accounts](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Transactions](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Messages](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Modules](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Protobuf](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Multistore and Keepers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[BaseApp](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Queries](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Events](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Context](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Testing](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Relaying with IBC](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Interchain Security](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Bridges](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Migrations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 1 Quiz](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 2 - First Steps](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[First Steps](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Setup Your Work Environment](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Run a Node, API, and CLI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Ignite CLI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Exercise - Make a Checkers Blockchain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Store Object](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Create and Save a Game Properly](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Add a Way to Make a Move](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Emit Game Information](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Record the Game Winner](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 2 Exercise](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[What is IBC?](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC/TAO - Connections (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC/TAO - Channels (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC/TAO - Clients (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC Token Transfer](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Interchain Accounts (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC Tooling](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[What is CosmJS?](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Your First CosmJS Actions](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Compose Complex Transactions](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Learn to Integrate Keplr](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Create Custom CosmJS Interfaces](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Put Your Games in Order](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Auto-Expiring Games](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Let Players Set a Wager](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Handle wager payments](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Integration tests](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Incentivize Players](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Help Find a Correct Move](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Play With Cross-Chain Tokens](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Understand IBC Denoms](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Go Relayer](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Hermes Relayer](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Create Custom Objects](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Create Custom Messages](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Get an External GUI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Integrate CosmJS and Keplr](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Backend Script for Game Indexing](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[IBC Application Developer Introduction](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Make a Module IBC-Enabled](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Adding Packet and Acknowledgment Data](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Create a Leaderboard Chain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Run in Production](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Prepare the Software to Run](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Prepare a Validator and Keys](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Prepare Where the Node Starts](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Prepare and Connect to Other Nodes](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Prepare and Do Migrations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Simulate Production in Docker](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Tally Player Info After Production](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Add a Leaderboard as a Module](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Simulate a Migration in Docker](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Final Exam](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[What's Next?](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

[Continue Your Interchain Journey](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html)

Docs Version Switcher

On this page

[Target setup](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#target-setup)

[Docker elements](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#docker-elements)

[The node image](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-node-image)

[The key manager image](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-key-manager-image)

[Blockchain elements](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#blockchain-elements)

[Basic initialization](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#basic-initialization)

[Keys](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#keys)

[Validator operator keys](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#validator-operator-keys)

[Alice's consensus key on the KMS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#alice-s-consensus-key-on-the-kms)

[Prepare the KMS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#prepare-the-kms)

[Import the consensus key](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#import-the-consensus-key)

[Set up the KMS connection](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-the-kms-connection)

[Genesis](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis)

[Set up chain ID](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-chain-id)

[Initial balances](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#initial-balances)

[Initial stakes](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#initial-stakes)

[Genesis assembly](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis-assembly)

[Genesis distribution](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis-distribution)

[Network preparation](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#network-preparation)

[Set up Alice's sentry](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-alice-s-sentry)

[Other nodes](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#other-nodes)

[Open Carol's node](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#open-carol-s-node)

[CORS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#cors)

[Compose elements](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#compose-elements)

[The executables that run](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-executables-that-run)

[The data each container needs](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-data-each-container-needs)

[The networks they run in](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-networks-they-run-in)

[Additional settings](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#additional-settings)

[Launching Compose](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#launching-compose)

[Interacting with Compose](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#interacting-with-compose)

[Stopping Compose](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#stopping-compose)

[Self-contained checkers blockchain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#self-contained-checkers-blockchain)

[CosmJS faucet](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#cosmjs-faucet)

[Dockerfile construction, checkers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#dockerfile-construction-checkers)

[Dockerfile construction, faucet](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#dockerfile-construction-faucet)

[Build your standalone images](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#build-your-standalone-images)

[Run your standalone checkers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#run-your-standalone-checkers)

[Test your standalone checkers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#test-your-standalone-checkers)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#simulate-production-in-docker) **Simulate Production in Docker**



Make sure you have everything you need before proceeding:

* You understand the concepts of [production](https://ida.interchain.io/tutorials/9-path-to-prod/).
* Docker is installed and you [understand it](https://ida.interchain.io/tutorials/5-docker-intro/).
* You have the checkers **Go** blockchain codebase with the CosmJS elements. If not, follow the [previous steps](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/5-server-side.html) or check out the [relevant version (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/tree/cosmjs-elements).
* You have the checkers **CosmJS** codebase. If not, follow the [previous steps](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/5-server-side.html) or checkout the [relevant version (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/tree/server-indexing).



In this section, you will:

* Prepare Docker elements.
* Handle keys.
* Prepare blockchain nodes.
* Prepare a blockchain genesis.
* Compose the lot in one orchestrated ensemble.
* Test it.

Before you launch yourself fully into production, it would be interesting to simulate a set of independent nodes on your computer. This can be prepared and orchestrated with [Docker Compose (opens new window)↗](https://docs.docker.com/get-started/08_using_compose).

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#target-setup) Target setup

In order to mimic a real setup, find an objective on which to focus. Here the objective is:

* Three independent parties - Alice, Bob, and Carol.
* Two independent validator nodes, run by Alice and Bob respectively, that can only communicate with their own sentries and do not expose RPC endpoints.
* Additionally, Alice's validator node uses Tendermint Key Management System (TMKMS) on a separate machine.
* The two sentry nodes, run by Alice and Bob, expose endpoints to the *world*.
* A regular node, run by Carol, that can communicate with the *world* and exposes endpoints for use by clients.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#docker-elements) Docker elements

Before looking at the specific Compose elements, you need to define what the *regular* Docker elements are.

You will run containers. You can start by giving them meaningful names:

* Alice's containers: sentry-alice, val-alice, and kms-alice.
* Bob's containers: sentry-bob and val-bob.
* Carol's container: node-carol.

Docker lets you simulate private networks. To meaningfully achieve the above target setup in terms of network separation, you use Docker's user-defined networks. This means:

* Alice's validator and key management system (KMS) are on their private network: name it net-alice-kms.
* Alice's validator and sentry are on their private network: name it net-alice.
* Bob's validator and sentry are on their private network: name it net-bob.
* There is a public network, i.e. *the world*, on which both sentries and Carol's node run: name it net-public.

Although every machine on the network is a bit different, in terms of Docker images there are only two image types:

1. The Tendermint nodes (validators, sentries, and regular nodes) will run checkersd within containers created from a single Docker image.
2. The Tendermint KMS node will run TMKMS from a different Docker image.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-node-image) The node image

The node image contains, and runs by default, the checkers executable. You first have to compile it, and then build the image.

First, build the executable(s) that will be launched by Docker Compose within the Docker images. Depending on your platform, you will use checkersd-linux-amd64 or checkersd-linux-arm64.

Update your Makefile with:



Copy

build-linux:

GOOS=linux GOARCH=amd64 go build -o ./build/checkersd-linux-amd64 ./cmd/checkersd/main.go

GOOS=linux GOARCH=arm64 go build -o ./build/checkersd-linux-arm64 ./cmd/checkersd/main.go

do-checksum-linux:

cd build && sha256sum \

checkersd-linux-amd64 checkersd-linux-arm64 \

> checkers-checksum-linux

build-linux-with-checksum: build-linux do-checksum-linux

build-darwin:

GOOS=darwin GOARCH=amd64 go build -o ./build/checkersd-darwin-amd64 ./cmd/checkersd/main.go

GOOS=darwin GOARCH=arm64 go build -o ./build/checkersd-darwin-arm64 ./cmd/checkersd/main.go

build-all: build-linux build-darwin

do-checksum-darwin:

cd build && sha256sum \

checkersd-darwin-amd64 checkersd-darwin-arm64 \

> checkers-checksum-darwin

build-darwin-with-checksum: build-all-darwin do-checksum-darwin

build-with-checksum: build-linux-with-checksum build-darwin-with-checksum

Makefile

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/Makefile" \l "L33-L45" \t "_blank)

If you have a CPU architecture that is neither amd64 nor arm64, update your Makefile accordingly.



If you copy-pasted directly into Makefile, do not forget to convert the spaces into tabs.

Now run either command:

**Local**

**With checkers\_i**

**With golang:1.18.7**



Copy

$ make build-with-checksum

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

make build-with-checksum

Use this command if you already built this checkers\_i image, as this will take less time.

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

golang:1.18.7 \

make build-with-checksum

Use this command if you did not already have the checkers\_i image. This command may take longer but should always work.

Now include the relevant executable inside your **production** image. You need to use a Debian/Ubuntu base image because you compiled on one in the previous step. Create a new Dockerfile-checkersd-debian with:



Copy

FROM --platform=linux debian:11-slim

ARG BUILDARCH

ENV LOCAL=/usr/local

COPY build/checkersd-linux-${BUILDARCH} ${LOCAL}/bin/checkersd

ENTRYPOINT [ "checkersd" ]

prod-sim /

Dockerfile-checkersd-debian

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/Dockerfile-checkersd-debian" \t "_blank)

Build the image with:



Copy

$ docker build -f prod-sim/Dockerfile-checkersd-debian . -t checkersd\_i

****

**Troubleshooting**

Depending on your installed version of Docker, you may have to add the flags:

Copy

--build-arg BUILDARCH=amd64

Or just manually replace ${BUILDARCH} with amd64 or whichever is your architecture.

****

**With alpine**

Because you want to simulate production, you can make the case that you prefer to use the smaller alpine Docker image. Alpine and Debian use different C compilers with different **dynamically**-linked C library dependencies. This makes their compiled executables incompatible – at least with the go build commands as they are declared in the Makefile.

In this case, you have the choice between:

1. Compiling from within alpine, using a [multi-stage Docker build (opens new window)](https://docs.docker.com/build/building/multi-stage/).

Copy

FROM --platform=linux golang:1.18.7-alpine AS builder

RUN apk update

RUN apk add make

WORKDIR /original

ADD . /original

RUN go build -o ./build/checkersd ./cmd/checkersd/main.go

FROM --platform=linux alpine

ENV LOCAL=/usr/local

COPY --from=builder /original/build/checkersd ${LOCAL}/bin/checkersd

ENTRYPOINT [ "checkersd" ]

prod-sim

Dockerfile-checkersd-alpine

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/Dockerfile-checkersd-alpine" \t "_blank)

Then building the image with:

Copy

$ docker build -f prod-sim/Dockerfile-checkersd-alpine . -t checkersd\_i

1. Instructing the compiler to link the C libraries **statically** with the use of the CGO\_ENABLED=0 [option (opens new window)](https://medium.com/pragmatic-programmers/compiling-your-go-application-for-co-ntainers-b513190471aa) in go build, or even in your Makefile:

Copy

build-linux:

- GOOS=linux GOARCH=amd64 go build -o ./build/checkersd-linux-amd64 ./cmd/checkersd/main.go

- GOOS=linux GOARCH=arm64 go build -o ./build/checkersd-linux-arm64 ./cmd/checkersd/main.go

+ CGO\_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o ./build/checkersd-linux-amd64 ./cmd/checkersd/main.go

+ CGO\_ENABLED=0 GOOS=linux GOARCH=arm64 go build -o ./build/checkersd-linux-arm64 ./cmd/checkersd/main.go

Then run make build-with-checksum again and use alpine in a new Dockerfile:

Copy

FROM --platform=linux alpine

ARG BUILDARCH

ENV LOCAL=/usr/local

COPY build/checkersd-linux-${BUILDARCH} ${LOCAL}/bin/checkersd

ENTRYPOINT [ "checkersd" ]

For maximum portability of your executables, you may in fact want to add CGO\_ENABLED=0 to all your go build commands.

Now you can run it:



Copy

$ docker run --rm -it checkersd\_i help

You should see a recognizable list of commands.

Each Docker container will run checkersd as root, which does not matter because it all happens in a container. Therefore, there is no need to create a specific additional user like you would in a serious production setting. For the same reason, there is also no need to create a service to launch it.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-key-manager-image) The key manager image

Alice runs the [Tendermint Key Management System (opens new window)↗](https://github.com/iqlusioninc/tmkms) on a separate machine. You need to prepare its image. The image will contain the executable, which you have to compile from its Rust code.

There are several considerations to clarify:

* How will you build it?
* What device will store the key?
* What KMS version works with your node version?

The **build step** is a good opportunity to use a [multi-stage Docker build (opens new window)↗](https://docs.docker.com/build/building/multi-stage/). With this technique:

1. You define a disposable image (the first stage) that clones the code and compiles it. This involves the download of Rust crates (i.e. packages). This image ends up being large but is then disposed of.
2. You define a slim image (the second stage) in which you only copy the compiled file. This is the image you keep for production. It ends up being small.

The disposable image needs to use Rust of at least version 1.56. Fortunately, there are ready-made Docker images. Pick [rust:1.64.0 (opens new window)↗](https://hub.docker.com/layers/library/rust/1.64.0/images/sha256-44ba8b8d8a2993694926cc847e1cce27937550c2e9eade4d9887ba90b2a2063f).

Next, the executable needs to be compiled for the specific **device** onto which your key will be stored. You do not use hardware keys in this setup. So, when building it, you use the [softsign extension (opens new window)↗](https://github.com/iqlusioninc/tmkms/blob/c56b496e1bb1187482d7a6fad23c4566329c951e/src/keyring/providers/softsign.rs). This is achieved by adding the flag --features=softsign.

Finally, what **version** of the TMKMS should you compile? A given TMKMS version can work with a limited set of specific Tendermint versions. Find the Tendermint version of your checkers code with:



Copy

$ grep tendermint/tendermint go.mod

It should return something like this:



Copy

github.com/tendermint/tendermint v0.34.19



Because here it is version 0.34, it is a good idea to use the KMS from [version 0.10.0 (opens new window)↗](https://github.com/iqlusioninc/tmkms/blob/main/CHANGELOG.md) upwards. At the time of writing, **version 0.12.2** still seems to support Tendermint v0.34. It is under the [v0.12.2 (opens new window)↗](https://github.com/iqlusioninc/tmkms/tree/v0.12.2) tag on Github. Pick this one.

Having collected the requisites, you can create the multi-staged Docker image in a new Dockerfile-ubuntu-tmkms:

**With debian**

**With alpine**



Copy

FROM --platform=linux rust:1.64.0 AS builder

RUN apt-get update

RUN apt-get install libusb-1.0-0-dev --yes

ENV LOCAL=/usr/local

ENV RUSTFLAGS=-Ctarget-feature=+aes,+ssse3

ENV TMKMS\_VERSION=v0.12.2

WORKDIR /root

RUN git clone --branch ${TMKMS\_VERSION} https://github.com/iqlusioninc/tmkms.git

WORKDIR /root/tmkms

RUN cargo build --release --features=softsign

# The production image starts here

FROM --platform=linux debian:11-slim

COPY --from=builder /root/tmkms/target/release/tmkms ${LOCAL}/bin

ENTRYPOINT [ "tmkms" ]

prod-sim /

Dockerfile-tmkms-debian

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/Dockerfile-tmkms-debian" \t "_blank)

Build the image as usual with:



Copy

$ docker build -f prod-sim/Dockerfile-tmkms-debian . -t tmkms\_i:v0.12.2

Copy

FROM --platform=linux rust:1.64.0-alpine AS builder

RUN apk update

RUN apk add libusb-dev=1.0.26-r0 musl-dev git

ENV LOCAL=/usr/local

ENV RUSTFLAGS=-Ctarget-feature=+aes,+ssse3

ENV TMKMS\_VERSION=v0.12.2

WORKDIR /root

RUN git clone --branch ${TMKMS\_VERSION} https://github.com/iqlusioninc/tmkms.git

WORKDIR /root/tmkms

RUN cargo build --release --features=softsign

# The production image starts here

FROM --platform=linux alpine

COPY --from=builder /root/tmkms/target/release/tmkms ${LOCAL}/bin

ENTRYPOINT [ "tmkms" ]

prod-sim

Dockerfile-tmkms-alpine

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/Dockerfile-tmkms-alpine" \t "_blank)

Build the image as usual with:

Copy

$ docker build -f prod-sim/Dockerfile-tmkms-alpine . -t tmkms\_i:v0.12.2

As you can see, the production stage is only three lines.

* If you built with Debian, the image should be about 90 MB.
* If you built with Alpine, the image should be about 20 MB.

Now run it:



Copy

$ docker run --rm -it tmkms\_i:v0.12.2

It returns you information about usage. You have just built the Tendermint Key Management System.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#blockchain-elements) Blockchain elements

Each container needs access to its private information, such as keys, genesis, and database. To facilitate data access and separation between containers, create folders that will map as a volume to the default /root/.checkers or /root/tmkms inside containers. One for each container:



Copy

$ mkdir -p prod-sim/kms-alice

$ mkdir -p prod-sim/node-carol

$ mkdir -p prod-sim/sentry-alice

$ mkdir -p prod-sim/sentry-bob

$ mkdir -p prod-sim/val-alice

$ mkdir -p prod-sim/val-bob

prod-sim

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/tree/run-prod/prod-sim" \t "_blank)

Also add the desktop computers of Alice and Bob, so that they never have to put keys on a server that should never see them:



Copy

$ mkdir -p prod-sim/desk-alice

$ mkdir -p prod-sim/desk-bob

prod-sim

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/tree/run-prod/prod-sim" \t "_blank)

For instance, when running a container for val-alice, you would create the volume mapping with a command like:



Copy

$ docker run ... \

-v $(pwd)/prod-sim/val-alice:/root/.checkers \

checkersd\_i ...

And for the KMS, like so:



Copy

$ docker run ... \

-v $(pwd)/prod-sim/kms-alice:/root/tmkms \

tmkms\_i:v0.12.2 ...

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#basic-initialization) Basic initialization

Before you can change the configuration you need to initialize it. Do it on all nodes with this one-liner:



Copy

$ echo -e desk-alice'\n'desk-bob'\n'node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

docker run --rm -i \

-v $(pwd)/prod-sim/{}:/root/.checkers \

checkersd\_i \

init checkers

As a secondary effect, this also creates the first shot of config/genesis.json on every node, although you will start work with the one on desk-alice.

Early decisions that you can make at this stage are:

* Deciding that the chain will be named checkers-1. It is a convention to append a number in case it has to go through a hard fork.
* Deciding that the staking denomination will be called upawn, which will be understood as 1 PAWN equals 1 million of upawn.



Do you need that many decimals? Yes and no. Depending on your version of the Cosmos SDK, there is a hard-coded value of base tokens that a validator has to stake, and the number is 10,000,000. If you do not have enough decimals, the *human* token would have to have a lot of zeroes.

The default initialization sets the base token to stake, so to get it to be upawn you need to make some changes:

1. In the authoritative [config/genesis.json (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/desk-alice/config/genesis.json#L63) (desk-alice's):



Copy

$ docker run --rm -it \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-i 's/"stake"/"upawn"/g' /root/.checkers/config/genesis.json

Note how the command overrides the default checkersd entry point and replaces it with --entrypoint sed.

1. In all seven [config/app.toml (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/desk-alice/config/app.toml#L11):



Copy

$ echo -e desk-alice'\n'desk-bob'\n'node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

docker run --rm -i \

-v $(pwd)/prod-sim/{}:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/([0-9]+)stake/\1upawn/g' /root/.checkers/config/app.toml

Make sure that [config/client.toml (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/desk-alice/config/client.toml#L9) mentions checkers-1, the chain's name:



Copy

$ echo -e desk-alice'\n'desk-bob'\n'node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

docker run --rm -i \

-v $(pwd)/prod-sim/{}:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/^chain-id = .\*$/chain-id = "checkers-1"/g' \

/root/.checkers/config/client.toml

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#keys) Keys

Some keys are created automatically, like the [node keys (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/node-carol/config/node_key.json). For others, you have to create them yourself. You will create:

* The validator operator keys for Alice and Bob.
* The consensus keys, whether they stay on Bob's node or are kept inside Alice's KMS.

Start with the keys for the validators and Alice's KMS Tendermint key.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#validator-operator-keys) Validator operator keys

First, you need to create the two validators' operation keys. Such a key is not meant to stay on the validating node when it runs, it is meant to be used at certain junctures only (for instance, to stake on behalf of Alice or Bob, as from their respective desktop computers). So you are going to create them by running "desktop" containers:

1. Use the --keyring-backend file.
2. Keep them in the mapped volume with --keyring-dir /root/.checkers/keys.

Create on desk-alice the operator key for val-alice:



Copy

$ docker run --rm -it \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

checkersd\_i \

keys \

--keyring-backend file --keyring-dir /root/.checkers/keys \

add alice

Use a passphrase you can remember. It does not need to be exceptionally complex as this is all a local simulation. This exercise uses password and stores this detail on file, which will become handy.



Copy

$ echo -n password > prod-sim/desk-alice/keys/passphrase.txt

prod-sim /

desk-alice /

keys /

passphrase.txt

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/desk-alice/keys/passphrase.txt" \t "_blank)

Because with this prod simulation you care less about safety, so much less in fact, you can even keep the mnemonic on file too.

Do the same for val-bob:



Copy

$ docker run --rm -it \

-v $(pwd)/prod-sim/desk-bob:/root/.checkers \

checkersd\_i \

keys \

--keyring-backend file --keyring-dir /root/.checkers/keys \

add bob

$ echo -n password > prod-sim/desk-bob/keys/passphrase.txt

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#alice-s-consensus-key-on-the-kms) Alice's consensus key on the KMS

To get the KMS to work, you have to:

* Prepare the KMS.
* Import Alice's consensus key into the KMS' softsign *device*.
* Have the KMS and the node talk to each other.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#prepare-the-kms) Prepare the KMS

As per [the documentation (opens new window)↗](https://github.com/iqlusioninc/tmkms/tree/v0.12.2#configuration-tmkms-init), initialize the KMS folder:



Copy

$ docker run --rm -it \

-v $(pwd)/prod-sim/kms-alice:/root/tmkms \

tmkms\_i:v0.12.2 \

init /root/tmkms

In the newly-created kms-alice/tmkms.toml file:

1. Make sure that you use the right protocol version. In your case:

**TOML**

**One-liner**



Copy

[[validator]]

...

protocol\_version = "v0.34"

prod-sim /

kms-alice /

tmkms.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/kms-alice/tmkms.toml" \l "L27" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/kms-alice:/root/tmkms \

--entrypoint sed \

tmkms\_i:v0.12.2 \

-Ei 's/^protocol\_version = .\*$/protocol\_version = "v0.34"/g' \

/root/tmkms/tmkms.toml

1. Pick an expressive name for the file that will contain the softsign key for val-alice:

**TOML**

**One-liner**



Copy

[[providers.softsign]]

...

path = "/root/tmkms/secrets/val-alice-consensus.key"

prod-sim /

kms-alice /

tmkms.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/kms-alice/tmkms.toml" \l "L19" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/kms-alice:/root/tmkms \

--entrypoint sed \

tmkms\_i:v0.12.2 \

-Ei 's/path = "\/root\/tmkms\/secrets\/cosmoshub-3-consensus.key"/path = "\/root\/tmkms\/secrets\/val-alice-consensus.key"/g' \

/root/tmkms/tmkms.toml

1. Replace cosmoshub-3 with checkers-1, the name of your blockchain, wherever the former appears:

**TOML**

**One-liner**



Copy

[[chain]]

id = "checkers-1"

...

[[providers.softsign]]

chain\_ids = ["checkers-1"]

...

[[validator]]

chain\_id = "checkers-1"

prod-sim /

kms-alice /

tmkms.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/kms-alice/tmkms.toml" \l "L24" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/kms-alice:/root/tmkms \

--entrypoint sed \

tmkms\_i:v0.12.2 \

-Ei 's/cosmoshub-3/checkers-1/g' /root/tmkms/tmkms.toml

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#import-the-consensus-key) Import the consensus key

Now you need to import val-alice's consensus key in secrets/val-alice-consensus.key.

The private key will no longer be needed on val-alice. However, during the genesis creation Alice will need access to her consensus public key. Save it in a new [pub\_validator\_key-val-alice.json (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/desk-alice/config/pub_validator_key-val-alice.json) on Alice's desk without any new line:



Copy

$ docker run --rm -t \

-v $(pwd)/prod-sim/val-alice:/root/.checkers \

checkersd\_i \

tendermint show-validator \

| tr -d '\n' | tr -d '\r' \

> prod-sim/desk-alice/config/pub\_validator\_key-val-alice.json

The consensus private key should not reside on the validator. You can simulate that by moving it out:



Copy

$ cp prod-sim/val-alice/config/priv\_validator\_key.json \

prod-sim/desk-alice/config/priv\_validator\_key-val-alice.json

$ mv prod-sim/val-alice/config/priv\_validator\_key.json \

prod-sim/kms-alice/secrets/priv\_validator\_key-val-alice.json

Import it into the softsign "device" as defined in [tmkms.toml (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/kms-alice/tmkms.toml#L19):



Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/kms-alice:/root/tmkms \

-w /root/tmkms \

tmkms\_i:v0.12.2 \

softsign import secrets/priv\_validator\_key-val-alice.json \

secrets/val-alice-consensus.key

On start, val-alice may still recreate a missing private key file due to how defaults are handled in the code. To prevent that, you can instead copy it from sentry-alice where it has no value.



Copy

$ cp prod-sim/sentry-alice/config/priv\_validator\_key.json \

prod-sim/val-alice/config/

With the key created you now set up the connection from kms-alice to val-alice.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-the-kms-connection) Set up the KMS connection

Choose a port unused on val-alice, for instance 26659, and inform kms-alice:

**TOML**

**One-liner**



Copy

[[validator]]

...

addr = "tcp://val-alice:26659"

prod-sim /

kms-alice /

tmkms.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/kms-alice/tmkms.toml" \l "L25" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/kms-alice:/root/tmkms \

--entrypoint sed \

tmkms\_i:v0.12.2 \

-Ei 's/^addr = "tcp:.\*$/addr = "tcp:\/\/val-alice:26659"/g' /root/tmkms/tmkms.toml

In the above, val-alice is the future network name of Alice's validator, and it will indeed be resolved to an IP address via Docker's internal DNS. In a real production setup, you would use a fully resolved IP address to avoid the vagaries of DNS.

Do not forget, you must inform Alice's validator that it should indeed listen on port 26659. In val-alice/config/config.toml:

**TOML**

**One-liner**



Copy

priv\_validator\_laddr = "tcp://0.0.0.0:26659"

prod-sim /

val-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/val-alice/config/config.toml" \l "L68" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/val-alice:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/priv\_validator\_laddr = ""/priv\_validator\_laddr = "tcp:\/\/0.0.0.0:26659"/g' \

/root/.checkers/config/config.toml



Make it listen on an IP address that is within the KMS private network.

0.0.0.0 represents all addresses of the node. In a real production setup, you would choose the IP address of the network card that is on the network common with kms-alice.

* Make sure it will not look for the consensus key on file:

**TOML**

**One-liner**



Copy

# priv\_validator\_key\_file = "config/priv\_validator\_key.json"

prod-sim /

val-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/val-alice/config/config.toml" \l "L61" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/val-alice:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/^priv\_validator\_key\_file/# priv\_validator\_key\_file/g' \

/root/.checkers/config/config.toml

* Make sure it will not look for the consensus state file either, as this is taken care of by the KMS:

**TOML**

**One-liner**



Copy

# priv\_validator\_state\_file = "data/priv\_validator\_state.json"

prod-sim /

val-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/val-alice/config/config.toml" \l "L64" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/val-alice:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/^priv\_validator\_state\_file/# priv\_validator\_state\_file/g' \

/root/.checkers/config/config.toml

Before moving on, make sure that the validator still has a priv\_validator\_key.json because the code may complain if the file cannot be found. You can copy the key from sentry-alice, which does not present any risk:



Copy

$ cp prod-sim/sentry-alice/config/priv\_validator\_key.json \

prod-sim/val-alice/config

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis) Genesis

With the keys in you can start fleshing out the genesis, which is already created.

You need to:

1. Set up the chain ID.
2. Add the initial balances.
3. Add the initial validator stakes.
4. Distribute the genesis file to all relevant nodes.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-chain-id) Set up chain ID

Earlier you chose checkers-1, so you adjust it here too:

**JSON**

**One-liner**



Copy

"chain\_id": "checkers-1",

prod-sim /

desk-alice /

config /

genesis.json

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/desk-alice/config/genesis.json" \l "L3" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/"chain\_id": "checkers"/"chain\_id": "checkers-1"/g' \

/root/.checkers/config/genesis.json

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#initial-balances) Initial balances

In this setup, Alice starts with 1,000 PAWN and Bob with 500 PAWN, of which Alice stakes 60 and Bob 40. With these amounts, the network cannot start if either of them is offline. Get their respective addresses:



Copy

$ ALICE=$(echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

checkersd\_i \

keys \

--keyring-backend file --keyring-dir /root/.checkers/keys \

show alice --address)

Replace password with the passphrase you picked when creating the keys.

Have Alice add her initial balance in the genesis:



Copy

$ docker run --rm -it \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

checkersd\_i \

add-genesis-account $ALICE 1000000000upawn

Now move the genesis file to desk-bob. This mimics what would happen in a real-life setup:



Copy

$ mv prod-sim/desk-alice/config/genesis.json \

prod-sim/desk-bob/config/

Have Bob add his own initial balance:



Copy

$ BOB=$(echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-bob:/root/.checkers \

checkersd\_i \

keys \

--keyring-backend file --keyring-dir /root/.checkers/keys \

show bob --address)

$ docker run --rm -it \

-v $(pwd)/prod-sim/desk-bob:/root/.checkers \

checkersd\_i \

add-genesis-account $BOB 500000000upawn

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#initial-stakes) Initial stakes

Alice and Bob both have initial stakes that they define via genesis transactions. You create them.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#bob-s-stake) Bob's stake

Bob is not using the Tendermint KMS but instead uses the validator key on file priv\_validator\_key.json. So, first make a copy of it on Bob's desktop.



Copy

$ cp prod-sim/val-bob/config/priv\_validator\_key.json \

prod-sim/desk-bob/config/priv\_validator\_key.json

Bob appears in second position in app\_state.accounts, so his account\_number ought to be 1; but it is in fact written as 0, so you use 0:



Copy

$ echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-bob:/root/.checkers \

checkersd\_i \

gentx bob 40000000upawn \

--keyring-backend file --keyring-dir /root/.checkers/keys \

--account-number 0 --sequence 0 \

--chain-id checkers-1 \

--gas 1000000 \

--gas-prices 0.1upawn

Again, insert Bob's chosen passphrase instead of password. Return the genesis to Alice:



Copy

$ mv prod-sim/desk-bob/config/genesis.json \

prod-sim/desk-alice/config/

It is Alice's turn to add her staking transaction.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#alice-s-stake) Alice's stake

Create Alice's genesis transaction using the specific validator public key that you saved on file, and not the key that would be taken from priv\_validator\_key.json by default (and is now missing):



Copy

$ echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

checkersd\_i \

gentx alice 60000000upawn \

--keyring-backend file --keyring-dir /root/.checkers/keys \

--account-number 0 --sequence 0 \

--pubkey $(cat prod-sim/desk-alice/config/pub\_validator\_key-val-alice.json) \

--chain-id checkers-1 \

--gas 1000000 \

--gas-prices 0.1upawn



It is useful to know this --pubkey method. If you were using a hardware key located on the KMS, this would be the canonical way of generating your genesis transaction.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis-assembly) Genesis assembly

With the two initial staking transactions created, have Alice include both of them in the genesis:



Copy

$ cp prod-sim/desk-bob/config/gentx/gentx-\* \

prod-sim/desk-alice/config/gentx

$ docker run --rm -it \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

checkersd\_i collect-gentxs

As an added precaution, confirm that it is a valid genesis:



Copy

$ docker run --rm -it \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

checkersd\_i \

validate-genesis

It should return:



Copy

File at /root/.checkers/config/genesis.json is a valid genesis file

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis-distribution) Genesis distribution

All the nodes that will run the executable need the final version of the genesis. Copy it across:

**Separate commands**

**One-liner**



Copy

$ cp prod-sim/desk-alice/config/genesis.json prod-sim/desk-bob/config

$ cp prod-sim/desk-alice/config/genesis.json prod-sim/node-carol/config

$ cp prod-sim/desk-alice/config/genesis.json prod-sim/sentry-alice/config

$ cp prod-sim/desk-alice/config/genesis.json prod-sim/sentry-bob/config

$ cp prod-sim/desk-alice/config/genesis.json prod-sim/val-alice/config

$ cp prod-sim/desk-alice/config/genesis.json prod-sim/val-bob/config

Copy

$ echo -e desk-bob'\n'node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

cp prod-sim/desk-alice/config/genesis.json prod-sim/{}/config

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#network-preparation) Network preparation

Because the validators are on a private network and fronted by sentries, you need to set up the configuration of each node so they can find each other; also to make sure that the sentries keep the validators' addresses private. What are the nodes' public keys? For instance, for val-alice, it is:



Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/val-alice:/root/.checkers \

checkersd\_i \

tendermint show-node-id

This returns something like:



Copy

f2673103417334a839f5c20096909c3023ba4903

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-alice-s-sentry) Set up Alice's sentry

The nodes that have access to val-alice should know Alice's sentry by this identifier:



Copy

f2673103417334a839f5c20096909c3023ba4903@val-alice:26656

Where:

* val-alice will be resolved via Docker's DNS.
* 26656 is the port as found in val-alice's configuration:



Copy

laddr = "tcp://0.0.0.0:26656"

prod-sim /

val-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/val-alice/config/config.toml" \l "L202" \t "_blank)

In the case of val-alice, only sentry-alice has access to it. Moreover, this is a persistent node. So you add it in sentry-alice's configuration:



Copy

persistent\_peers = "f2673103417334a839f5c20096909c3023ba4903@val-alice:26656"

prod-sim /

sentry-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/sentry-alice/config/config.toml" \l "L215" \t "_blank)

sentry-alice also has access to sentry-bob and node-carol, although these nodes should probably not be considered persistent. You will add them under "seeds". First, collect the same information from these nodes:



Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/sentry-bob:/root/.checkers \

checkersd\_i \

tendermint show-node-id

$ docker run --rm -i \

-v $(pwd)/prod-sim/node-carol:/root/.checkers \

checkersd\_i \

tendermint show-node-id

Eventually, in sentry-alice, you should have:



Copy

seeds = "7009cc51174dce87c31f537fe8fed906349a27f4@sentry-bob:26656,8f1bafad62a4a1f8678214d96a8b2ae2ed140cf7@node-carol:26656"

persistent\_peers = "f2673103417334a839f5c20096909c3023ba4903@val-alice:26656"

prod-sim /

sentry-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/sentry-alice/config/config.toml" \l "L212-L215" \t "_blank)

Before moving on to other nodes, remember that sentry-alice should keep val-alice secret. Set:



Copy

private\_peer\_ids = "f2673103417334a839f5c20096909c3023ba4903"

prod-sim /

sentry-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/sentry-alice/config/config.toml" \l "L261" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#other-nodes) Other nodes

Repeat the procedure for the other nodes, taking into account their specific circumstances:

* val-alice's:



Copy

persistent\_peers = "83144b58031953ad60eaccb0a790955450f1ddef@sentry-alice:26656"

prod-sim /

val-alice /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/val-alice/config/config.toml" \l "L215" \t "_blank)

* val-bob's:



Copy

persistent\_peers = "7009cc51174dce87c31f537fe8fed906349a27f4@sentry-bob:26656"

prod-sim /

val-bob /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/val-bob/config/config.toml" \l "L215" \t "_blank)

* sentry-bob's:



Copy

seeds = "83144b58031953ad60eaccb0a790955450f1ddef@sentry-alice:26656,8f1bafad62a4a1f8678214d96a8b2ae2ed140cf7@node-carol:26656"

persistent\_peers = "1e0d99ccf83b49e7aca852e82074c8e7f0e99d73@val-bob:26656"

private\_peer\_ids = "1e0d99ccf83b49e7aca852e82074c8e7f0e99d73"

prod-sim /

sentry-bob /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/sentry-bob/config/config.toml" \l "L212-L215" \t "_blank)

* node-carol's:



Copy

seeds = "83144b58031953ad60eaccb0a790955450f1ddef@sentry-alice:26656,7009cc51174dce87c31f537fe8fed906349a27f4@sentry-bob:26656"

prod-sim /

node-carol /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/node-carol/config/config.toml" \l "L212" \t "_blank)



For the avoidance of doubt, sentry-alice has a different address depending on which node resolves the address:

* When it is resolved from val-alice, the resolution takes place in net-alice.
* When it is resolved from sentry-bob, the resolution takes place in net-public.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#open-carol-s-node) Open Carol's node

Carol created her node to open it to the public. Make sure that her node's RPC listens on all IP addresses:

**TOML**

**One-liner**



Copy

laddr = "tcp://0.0.0.0:26657"

prod-sim /

node-carol /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/node-carol/config/config.toml" \l "L91" \t "_blank)

Copy

$ docker run --rm -i \

-v $(pwd)/prod-sim/node-carol:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei '0,/^laddr = .\*$/{s/^laddr = .\*$/laddr = "tcp:\/\/0.0.0.0:26657"/}' \

/root/.checkers/config/config.toml

Note that it has to replace only the first occurrence in the whole.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#cors) CORS

As a last step, you can disable CORS policies so that you are not surprised if you use a node from a Web browser.

1. In config.toml:

**TOML**

**One-liner**



Copy

cors\_allowed\_origins = ["\*"]

prod-sim /

node-carol /

config /

config.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/node-carol/config/config.toml" \l "L96" \t "_blank)

Copy

echo -e node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

docker run --rm -i \

-v $(pwd)/prod-sim/{}:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/^cors\_allowed\_origins = \[\]/cors\_allowed\_origins = \["\\*"\]/g' \

/root/.checkers/config/config.toml

1. In app.toml, first location:

**TOML**

**One-liner**



Copy

enabled-unsafe-cors = true

prod-sim /

node-carol /

config /

app.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/node-carol/config/app.toml" \l "L129" \t "_blank)

Copy

echo -e node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

docker run --rm -i \

-v $(pwd)/prod-sim/{}:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/^enabled-unsafe-cors = false/enabled-unsafe-cors = true/g' \

/root/.checkers/config/app.toml

1. In app.toml, second location:

**TOML**

**One-liner**



Copy

enable-unsafe-cors = true

prod-sim /

node-carol /

config /

app.toml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/node-carol/config/app.toml" \l "L181" \t "_blank)

Copy

echo -e node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

docker run --rm -i \

-v $(pwd)/prod-sim/{}:/root/.checkers \

--entrypoint sed \

checkersd\_i \

-Ei 's/^enable-unsafe-cors = false/enable-unsafe-cors = true/g' \

/root/.checkers/config/app.toml

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#compose-elements) Compose elements

You have prepared:

* The basic Docker elements
* The blockchain elements
* The network elements

Time to assemble them in Compose.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-executables-that-run) The executables that run

You define the different containers as services. Important elements to start with are:

* In container\_name, you use names that make them intelligible and match the names you used in the above preparations.
* In image, you declare the Docker image to use.
* In command, you define the command to use when launching the image.

In a new [prod-sim/docker-compose.yml (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/docker-compose.yml), write:



Copy

version: "3.7"

services:

kms-alice:

command: start --config /root/tmkms/tmkms.toml

container\_name: kms-alice

image: tmkms\_i:v0.12.2

val-alice:

command: start

container\_name: val-alice

image: checkersd\_i

sentry-alice:

command: start

container\_name: sentry-alice

image: checkersd\_i

val-bob:

command: start

container\_name: val-bob

image: checkersd\_i

sentry-bob:

command: start

container\_name: sentry-bob

image: checkersd\_i

node-carol:

command: start

container\_name: node-carol

image: checkersd\_i

Of course, Alice's and Bob's desktop computers are not part of the server infrastructure.

You are going to further refine the service definitions next, starting with the disk volumes.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-data-each-container-needs) The data each container needs

Each container needs to access its own private folder, prepared earlier, and only that folder. Declare the volume mappings with paths relative to the docker-compose.yml file:



Copy

services:

kms-alice:

...

volumes:

- ./kms-alice:/root/tmkms

val-alice:

...

volumes:

- ./val-alice:/root/.checkers

sentry-alice:

...

volumes:

- ./sentry-alice:/root/.checkers

val-bob:

...

volumes:

- ./val-bob:/root/.checkers

sentry-bob:

...

volumes:

- ./sentry-bob:/root/.checkers

node-carol:

...

volumes:

- ./node-carol:/root/.checkers

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-networks-they-run-in) The networks they run in

The user-defined networks need to mimic the desired separation of machines/containers, it can be self-explanatorily declared as:



Copy

networks:

net-alice-kms:

net-alice:

net-bob:

net-public:

prod-sim /

docker-compose.yml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/docker-compose.yml" \l "L3-L7" \t "_blank)

With the network declaration done, the associating of each computer to each network can be written as:



Copy

services:

kms-alice:

...

networks:

- net-alice-kms

val-alice:

...

networks:

- net-alice-kms

- net-alice

sentry-alice:

...

networks:

- net-alice

- net-public

val-bob:

...

networks:

- net-bob

sentry-bob:

...

networks:

- net-bob

- net-public

node-carol:

...

networks:

- net-public

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#additional-settings) Additional settings

The KMS connects to the node and can reconnect. So have val-alice start after kms-alice:



Copy

services:

val-alice:

...

depends\_on:

- kms-alice

prod-sim /

docker-compose.yml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/docker-compose.yml" \l "L25-L26" \t "_blank)

With all these computers on their Docker networks, you may still want to access one of them to query the blockchain, or to play games. In order to make your host computer look like an open node, expose Carol's node on all addresses of your host:



Copy

services:

node-carol:

...

ports:

- 0.0.0.0:26657:26657

prod-sim /

docker-compose.yml

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/docker-compose.yml" \l "L69-L70" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#launching-compose) Launching Compose



After this long preparation, before launch, it could be a good time to make a Git commit so that you can restore easily.

You are now ready to start your setup with a name other than the folder it is running in:



Copy

$ docker compose \

--file prod-sim/docker-compose.yml \

--project-name checkers-prod up \

--detach

At this point, it should be apparent that you need to update .gitignore. Add:



Copy

build/

prod-sim/\*/config/addrbook.json

prod-sim/\*/data/\*

!prod-sim/\*/data/priv\_validator\_state.json

.gitignore

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/.gitignore" \l "L4-L7" \t "_blank)

Note how priv\_validator\_state.json is necessary if you want to try again on another host, otherwise, it would be ignored by Git.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#interacting-with-compose) Interacting with Compose

Your six containers are running. To monitor their status, and confirm that they are running, use the provided Docker container interface.

Now you can connect to node-carol to start interacting with the blockchain as you would a normal node. For instance, to ask a simple status:

**In net-public**

**Local Mac**

**Windows Powershell**

**In plain Docker**



Copy

$ docker run --rm -it \

--network checkers-prod\_net-public \

checkersd\_i status \

--node "tcp://node-carol:26657"

Note how the net-public network name is prefixed with the Compose project name. If in doubt, you can run:



Copy

$ docker network ls

Copy

$ ./build/checkersd-darwin-amd64 status \

--node "tcp://localhost:26657"

Copy

> ./build/checkersd-windows-amd64 status \

--node "tcp://localhost:26657"

Copy

$ docker run --rm -it \

checkersd\_i status \

--node "tcp://192.168.0.2:26657"

Here you would replace 192.168.0.2 with the actual IP address of your host computer.

From this point on everything you already know how to do, such as connecting to your local node, applies.

Whenever you submit a transaction to node-carol, it will be propagated to the sentries and onward to the validators.

At this juncture, you may ask: Is it still possible to run a full game in almost a single block, as you did earlier in the [CosmJS integration tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#multiple-transactions-in-a-block)? After all, when node-carol passes on the transactions as they come, it is not certain that the recipients will honor the order in which they were received. Of course, they make sure to order Alice's transactions, thanks to the sequence, as well as Bob's. But do they keep the A-B-A-B... order in which they were sent?

To find out, you need to credit the tests' Alice and Bob accounts:

1. Get your prod setup's respective addresses for Alice and Bob:

**Alice**

**Bob**



Copy

$ alice=$(echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

checkersd\_i:v1-alpine \

keys \

--keyring-backend file --keyring-dir /root/.checkers/keys \

show alice --address)

Copy

$ bob=$(echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-bob:/root/.checkers \

checkersd\_i:v1-alpine \

keys \

--keyring-backend file --keyring-dir /root/.checkers/keys \

show bob --address)

1. The CosmJS tests use stake and token, whereas this production setup uses only upawn. Therefore, do a *text* search and change all occurrences of stake and token to upawn in [client/test/integration/stored-game-action.ts (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/main/test/integration/stored-game-action.ts). Also remove the [upawn: 1, (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/main/test/integration/stored-game-action.ts#L56-L60) lines that prevent compilation.
2. Credit the test accounts so that the CosmJS tests do not attempt to call a missing faucet:



Copy

$ echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-alice:/root/.checkers \

--network checkers-prod\_net-public \

checkersd\_i:v1-alpine tx bank \

send $alice cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql 300upawn \

--from $alice \

--keyring-backend file --keyring-dir /root/.checkers/keys \

--chain-id checkers-1 \

--node http://node-carol:26657 \

--broadcast-mode block --yes

$ echo password | docker run --rm -i \

-v $(pwd)/prod-sim/desk-bob:/root/.checkers \

--network checkers-prod\_net-public \

checkersd\_i:v1-alpine tx bank \

send $bob cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22 300upawn \

--from $bob \

--keyring-backend file --keyring-dir /root/.checkers/keys \

--chain-id checkers-1 \

--node http://node-carol:26657 \

--broadcast-mode block --yes

Now you can launch everything within net-public:



Copy

$ docker run --rm -it \

-v $(pwd)/client:/client -w /client \

--network checkers-prod\_net-public \

--env RPC\_URL="http://node-carol:26657" \

node:18.7-slim \

npm test

The tests should pass. *Should* as in there is no protocol guarantee that they will, but it looks like they do.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#stopping-compose) Stopping Compose

To stop your whole setup, run:



Copy

$ docker compose --project-name checkers-prod down

****

**Troubleshooting**

1

You may encounter an error such as:

Copy

Error: error during handshake: error on replay: validator set is nil in genesis and still empty after InitChain

If encountering this or other errors, you may want to do a state reset on all nodes:

Copy

$ echo -e node-carol'\n'sentry-alice'\n'sentry-bob'\n'val-alice'\n'val-bob \

| xargs -I {} \

docker run --rm -i \

-v $(pwd)/prod-sim/{}:/root/.checkers \

checkersd\_i \

tendermint unsafe-reset-all \

--home /root/.checkers

2

If one of your services (for example, sentry-bob) fails to start because it could not resolve one of the other containers, you can restart that service independently with:

Copy

$ docker compose restart sentry-bob

3

If you want to get more detailed errors from your KMS, you can add a flag in its service definition:

Copy

services:

kms-alice:

...

environment:

- RUST\_BACKTRACE=1

4

If you want to erase all states after a good run, and if you have a Git commit from which to restore the state files, you can create a [new script (opens new window)](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/prod-sim/unsafe-reset-state.sh) for that.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#self-contained-checkers-blockchain) Self-contained checkers blockchain

Now may be a good time to prepare a standalone setup that can be used by anyone who wants to test a checkers blockchain with minimum effort. The target setup ought to have the following characteristics:

* It uses a single Dockerfile.
* Such an image could be generated and uploaded into a Docker image registry to increase ease of use.
* It can be run by someone who just wants to try checkers without going through node and genesis setups.
* The Dockerfile does not need to be in the repository to be usable. It could be copied elsewhere and still work, i.e. no ADDing local files.
* The image(s) should be as small as is reasonable.
* It uses stake instead of upawn, and has token so as to be compatible with the current state of the checkers CosmJS exercise.
* It also provides a faucet to further facilitate tests.
* It sacrifices key safety to increase ease of use.

The CosmJS exercise already references this standalone Dockerfile, so this is a circular reference. You can still work on it on your own now.

It is possible to build [more than one Docker image (opens new window)↗](https://docs.docker.com/build/building/multi-stage/#stop-at-a-specific-build-stage) out of a single Dockerfile by using the multi-stage build mechanism.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#cosmjs-faucet) CosmJS faucet

When running ignite chain serve you also get a faucet, which was called when running the [CosmJS integration tests](https://ida.interchain.io/hands-on-exercise/3-cosmjs-adv/2-cosmjs-messages.html#adding-tests). In fact, CosmJS also offers a [faucet package (opens new window)↗](https://www.npmjs.com/package/@cosmjs/faucet). Its API differs from Ignite's faucet. If you went through the CosmJS exercise, you saw its API being called too.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#dockerfile-construction-checkers) Dockerfile construction, checkers

You assemble this multi-stage Dockerfile step by step, starting with the checkers part:

1

Build the checkers executable as you have learned in this section, but this time from the public repository so as to not depend on local files:



Copy

FROM --platform=linux golang:1.18.7-alpine AS builder

ENV CHECKERS\_VERSION=main

RUN apk add --update --no-cache make git

WORKDIR /root

RUN git clone --depth 1 --branch ${CHECKERS\_VERSION} https://github.com/cosmos/b9-checkers-academy-draft.git checkers

WORKDIR /root/checkers

RUN go build -o ./build/checkersd ./cmd/checkersd/main.go

FROM --platform=linux alpine

COPY --from=builder /root/checkers/build/checkersd /usr/local/bin/checkersd

2

To offer maximum determinism, you are going to reuse unprotected keys. First you need to create them with checkersd separately, somewhere *unimportant* like a temporary container:



Copy

$ checkersd keys add alice --keyring-backend test

This returns something like:



Copy

- name: alice

type: local

address: cosmos1am3fnp5dd6nndk5jyjq9mpqh3yvt2jmmdv83xn

pubkey: '{"@type":"/cosmos.crypto.secp256k1.PubKey","key":"A/E6dHn3W2XvCrLkhp/dNxAQyVpmduxEXPBg/nP/PyMa"}'

mnemonic: ""

\*\*Important\*\* write this mnemonic phrase in a safe place.

It is the only way to recover your account if you ever forget your password.

zebra burden afford work power afraid field creek laugh govern upgrade project glue ceiling lounge mobile romance pear relief either panel expect eagle jacket

Make a note of the mnemonic, so as to reuse it in the faucet's definition.

Add this new address information:



Copy

COPY --from=builder /root/checkers/build/checkersd /usr/local/bin/checkersd

+ ENV ALICE=cosmos1am3fnp5dd6nndk5jyjq9mpqh3yvt2jmmdv83xn

3

You want to import the only validator's private key into the Docker image's own test keyring. To export it first, run:



Copy

$ echo password | checkersd keys export alice --keyring-backend test

This returns something like:



Copy

-----BEGIN TENDERMINT PRIVATE KEY-----

kdf: bcrypt

salt: BE58E64E619563E337C6D899F06BF022

type: secp256k1

fOM6ZNruv1AirP/KYq1ZdMLdp8ynpk4cGPCsNThmEvRvqkhONpo9S1+tw6/WiFRN

+ZZrSaeqYO/3JKIr4cKRsaD460vtoK53crvJ/bE=

=1EBo

-----END TENDERMINT PRIVATE KEY-----

Next, add it as a local file in Dockerfile:



Copy

ENV ALICE=cosmos1am3fnp5dd6nndk5jyjq9mpqh3yvt2jmmdv83xn

+ RUN mkdir -p /root/.checkers/keys

+ # This private key file corresponds to the mnemonic above

+ RUN echo -----BEGIN TENDERMINT PRIVATE KEY----- > /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo kdf: bcrypt >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo salt: A67D88136A462383A2DD30727510DF59 >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo type: secp256k1 >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo 7gttUkxkpWxlI9tF0/vEYHvysmKTc/mG/aZ8dMF3u7a8xkPgVLa/Z75k/46nr0yN >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo FP/h5zTVYoP8tMnvVLV0koVAOV4QQurD5C7l3N8= >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo =qyP6 >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo -----END TENDERMINT PRIVATE KEY----- >> /root/.checkers/keys/encrypted-private-key-alice.txt

Now import it:



Copy

RUN echo -----END TENDERMINT PRIVATE KEY----- >> /root/.checkers/keys/encrypted-private-key-alice.txt

+ RUN echo password | checkersd keys import alice /root/.checkers/keys/encrypted-private-key-alice.txt --keyring-backend test

4

You then create a genesis as you learned in this section, while making sure the other configuration files are also configured permissively:



Copy

RUN echo password | checkersd keys import alice /root/.checkers/keys/encrypted-private-key-alice.txt --keyring-backend test

+ RUN checkersd init checkers

+ RUN sed -Ei 's/^enable-unsafe-cors = false/enable-unsafe-cors = true/g' /root/.checkers/config/app.toml

+ RUN sed -Ei 's/^enabled-unsafe-cors = false/enabled-unsafe-cors = true/g' /root/.checkers/config/app.toml

+ RUN sed -Ei 's/^laddr = "tcp:\/\/127.0.0.1:26657"/laddr = "tcp:\/\/0.0.0.0:26657"/g' /root/.checkers/config/config.toml

+ RUN sed -Ei 's/^cors\_allowed\_origins = \[\]/cors\_allowed\_origins = \["\\*"\]/g' /root/.checkers/config/config.toml

+ RUN sed -Ei 's/^chain-id = .\*$/chain-id = "checkers-1"/g' /root/.checkers/config/client.toml

+ RUN sed -Ei 's/"chain\_id": "checkers"/"chain\_id": "checkers-1"/g' /root/.checkers/config/genesis.json

+ RUN checkersd add-genesis-account $ALICE 1000000000000000stake,1000000000000000token

+ RUN checkersd gentx alice 10000000stake --keyring-backend test \

+ --account-number 0 --sequence 0 --chain-id checkers-1 \

+ --gas 1000000 --gas-prices 0.1stake

+ RUN checkersd collect-gentxs

Note that:

* You add both stake and token to Alice, so as to increase compatibility with the CosmJS exercise.
* The genesis transaction is only with stake.

5

Finally, advertise the gRPC port so that any user knows they ought to forward to the host, and have checkersd start by default:



Copy

RUN checkersd collect-gentxs

+ EXPOSE 26657

+ ENTRYPOINT [ "checkersd" ]

Your standalone checkers blockchain with a single validator is ready.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#dockerfile-construction-faucet) Dockerfile construction, faucet

Moving on to the faucet, you continue adding to the same Dockerfile.

1

You start its definition as a separate independent stage:



Copy

FROM --platform=linux node:18.7-alpine AS cosmos-faucet

Install the CosmJS faucet package:



Copy

FROM --platform=linux node:18.7-alpine AS cosmos-faucet

+ ENV COSMJS\_VERSION=0.28.11

+ RUN npm install @cosmjs/faucet@${COSMJS\_VERSION} --global --production

2

Configure the faucet:



Copy

RUN npm install @cosmjs/faucet@${COSMJS\_VERSION} --global --production

+ ENV FAUCET\_CONCURRENCY=2

+ ENV FAUCET\_PORT=4500

+ ENV FAUCET\_GAS\_PRICE=0.001stake

+ ENV FAUCET\_MNEMONIC="zebra burden afford work power afraid field creek laugh govern upgrade project glue ceiling lounge mobile romance pear relief either panel expect eagle jacket"

+ ENV FAUCET\_ADDRESS\_PREFIX=cosmos

+ ENV FAUCET\_TOKENS="stake, token"

+ ENV FAUCET\_CREDIT\_AMOUNT\_STAKE=100

+ ENV FAUCET\_CREDIT\_AMOUNT\_TOKEN=100

+ ENV FAUCET\_COOLDOWN\_TIME=0



Be aware:

* A concurrency of at least 2 is necessary for the CosmJS exercise, because when crediting in before it launches two simultaneous requests. The faucet does **not** internally keep track of the accounts' sequences and instead uses its *distributor* accounts in a round-robin fashion.
* You used port 4500 to mimic that of Ignite's faucet, so as to be conveniently compatible with the CosmJS exercise.
* You pasted the mnemonic that you obtained in the previous key-creation steps.
* You reused the token denominations as found in the CosmJS exercise.

3

Finish the faucet declaration with the port to share and the default command to launch:



Copy

ENV FAUCET\_COOLDOWN\_TIME=0

+ EXPOSE 4500

+ ENTRYPOINT [ "cosmos-faucet" ]

You now have a complete setup definition. Time to build the images.

****

**Complete Dockerfile**

Copy

# Faucet: docker build --target faucet

FROM --platform=linux node:18.7-alpine AS cosmos-faucet

ENV COSMJS\_VERSION=0.28.11

RUN npm install @cosmjs/faucet@${COSMJS\_VERSION} --global --production

ENV FAUCET\_CONCURRENCY=2

ENV FAUCET\_PORT=4500

ENV FAUCET\_GAS\_PRICE=0.001stake

# Prepared keys for determinism

ENV FAUCET\_MNEMONIC="zebra burden afford work power afraid field creek laugh govern upgrade project glue ceiling lounge mobile romance pear relief either panel expect eagle jacket"

ENV FAUCET\_ADDRESS\_PREFIX=cosmos

ENV FAUCET\_TOKENS="stake, token"

ENV FAUCET\_CREDIT\_AMOUNT\_STAKE=100

ENV FAUCET\_CREDIT\_AMOUNT\_TOKEN=100

ENV FAUCET\_COOLDOWN\_TIME=0

EXPOSE 4500

ENTRYPOINT [ "cosmos-faucet" ]

# Checkersd builder

FROM --platform=linux golang:1.18.7-alpine AS builder

ENV CHECKERS\_VERSION=main

RUN apk add --update --no-cache make git

WORKDIR /root

RUN git clone --depth 1 --branch ${CHECKERS\_VERSION} https://github.com/cosmos/b9-checkers-academy-draft.git checkers

WORKDIR /root/checkers

RUN go build -o ./build/checkersd ./cmd/checkersd/main.go

# Checkersd in production

FROM --platform=linux alpine

COPY --from=builder /root/checkers/build/checkersd /usr/local/bin/checkersd

# This address corresponds to the mnemonic above

ENV ALICE=cosmos1am3fnp5dd6nndk5jyjq9mpqh3yvt2jmmdv83xn

RUN mkdir -p /root/.checkers/keys

# This private key file corresponds to the mnemonic above

RUN echo -----BEGIN TENDERMINT PRIVATE KEY----- > /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo kdf: bcrypt >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo salt: A67D88136A462383A2DD30727510DF59 >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo type: secp256k1 >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo 7gttUkxkpWxlI9tF0/vEYHvysmKTc/mG/aZ8dMF3u7a8xkPgVLa/Z75k/46nr0yN >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo FP/h5zTVYoP8tMnvVLV0koVAOV4QQurD5C7l3N8= >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo =qyP6 >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo -----END TENDERMINT PRIVATE KEY----- >> /root/.checkers/keys/encrypted-private-key-alice.txt

RUN echo password | checkersd keys import alice /root/.checkers/keys/encrypted-private-key-alice.txt --keyring-backend test

RUN checkersd init checkers

RUN sed -Ei 's/^enable-unsafe-cors = false/enable-unsafe-cors = true/g' /root/.checkers/config/app.toml

RUN sed -Ei 's/^enabled-unsafe-cors = false/enabled-unsafe-cors = true/g' /root/.checkers/config/app.toml

RUN sed -Ei 's/^laddr = "tcp:\/\/127.0.0.1:26657"/laddr = "tcp:\/\/0.0.0.0:26657"/g' /root/.checkers/config/config.toml

RUN sed -Ei 's/^cors\_allowed\_origins = \[\]/cors\_allowed\_origins = \["\\*"\]/g' /root/.checkers/config/config.toml

RUN sed -Ei 's/^chain-id = .\*$/chain-id = "checkers-1"/g' /root/.checkers/config/client.toml

RUN sed -Ei 's/"chain\_id": "checkers"/"chain\_id": "checkers-1"/g' /root/.checkers/config/genesis.json

RUN checkersd add-genesis-account $ALICE 1000000000000000stake,1000000000000000token

RUN checkersd gentx alice 10000000stake --keyring-backend test \

--account-number 0 --sequence 0 --chain-id checkers-1 \

--gas 1000000 --gas-prices 0.1stake

RUN checkersd collect-gentxs

EXPOSE 26657

ENTRYPOINT [ "checkersd" ]

Dockerfile-standalone

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/Dockerfile-standalone" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#build-your-standalone-images) Build your standalone images

Your two Docker images will run in the same Docker network. To build them, run:

**Checkers**

**Faucet**



Copy

$ docker build . \

-f Dockerfile-standalone \

-t checkersd\_i:standalone

Copy

$ docker build . \

-f Dockerfile-standalone \

--target cosmos-faucet \

-t cosmos-faucet\_i:0.28.11

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#run-your-standalone-checkers) Run your standalone checkers

With your images built, you can launch both checkersd and the faucet process right away. Allow for a few seconds between them:

**Checkers**

**Faucet**



Copy

$ docker network create checkers-net

$ docker run --rm -it \

-p 26657:26657 \

--name checkers \

--network checkers-net \

--detach \

checkersd\_i:standalone start

Checkers needs about **10 seconds** to become operational. Wait that long before launching the faucet.

If your checkers-net network already exists, the first command fails with:



Copy

Error response from daemon: network with name checkers-net already exists

But that is okay.

Copy

$ docker run --rm -it \

-p 4500:4500 \

--name cosmos-faucet \

--network checkers-net \

--detach \

cosmos-faucet\_i:0.28.11 start http://checkers:26657

The faucet needs about **20 seconds** to become operational, as it sends four transactions to its distributor accounts. Wait that long before launching any CosmJS tests.



Be aware:

* Both processes are started as --detached, which is how they are typically started by users who do not care about the details. If you get errors then stop, remove this flag, and restart to see the logs.
* Checkers is started with --name checkers, whose name is then reused in the node address http://checkers:26657 when launching the faucet.

On a side-note, if you want to access Alice's address in order to access her balance, you can run:



Copy

$ docker exec -it checkers \

sh -c "checkersd query bank balances \$ALICE"

And to check the faucet status, you can use:



Copy

$ curl http://localhost:4500/status

You now have a container running both the checkers and a faucet. You are ready to run your CosmJS tests in client.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#test-your-standalone-checkers) Test your standalone checkers

Which RPC\_URL and which FAUCET\_URL will the tests require?

* If locally, the defaults will do:



Copy

RPC\_URL="http://localhost:26657"

FAUCET\_URL="http://localhost:4500"

* If running from Docker, you have to pass values to the tests that resolve via Docker's automatic name resolution within checkers-net:



Copy

RPC\_URL="http://checkers:26657"

FAUCET\_URL="http://cosmos-faucet:4500"

Now you can launch the tests:

**Local**

**Docker**



Copy

$ npm test --prefix client

Copy

$ docker run --rm -it \

-v $(pwd)/client:/client -w /client \

--network checkers-net \

--env RPC\_URL="http://checkers:26657" \

--env FAUCET\_URL="http://cosmos-faucet:4500" \

node:18.7-slim \

npm test

To stop (and --rm) both containers, run:



Copy

$ docker stop cosmos-faucet checkers

$ docker network rm checkers-net

synopsis

To summarize, this section has explored:

* How to prepare Docker images.
* How to prepare nodes for a simulated production setup.
* How to prepare a Tendermint Key Management System for a simulated production setup.
* How to prepare a blockchain genesis with multiple parties.
* How to launch all that with the help of Docker Compose.
* How to create a standalone blockchain server in a container.
* How to create a small faucet to assist with running your CosmJS integration tests.

previous

[](https://ida.interchain.io/tutorials/9-path-to-prod/7-migration.html)

**[Prepare and Do Migrations](https://ida.interchain.io/tutorials/9-path-to-prod/7-migration.html)**

up next

**[Tally Player Info After Production](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)**

[[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

Rate this Page

icon smile

icon meh

icon frown

Would you like to add a message?

Submit

Thank you for your Feedback!

[](https://ida.interchain.io/ida-course/discord-info.html)

On this page

[Target setup](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#target-setup)

[Docker elements](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#docker-elements)

[The node image](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-node-image)

[The key manager image](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-key-manager-image)

[Blockchain elements](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#blockchain-elements)

[Basic initialization](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#basic-initialization)

[Keys](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#keys)

[Validator operator keys](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#validator-operator-keys)

[Alice's consensus key on the KMS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#alice-s-consensus-key-on-the-kms)

[Prepare the KMS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#prepare-the-kms)

[Import the consensus key](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#import-the-consensus-key)

[Set up the KMS connection](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-the-kms-connection)

[Genesis](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis)

[Set up chain ID](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-chain-id)

[Initial balances](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#initial-balances)

[Initial stakes](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#initial-stakes)

[Genesis assembly](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis-assembly)

[Genesis distribution](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#genesis-distribution)

[Network preparation](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#network-preparation)

[Set up Alice's sentry](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#set-up-alice-s-sentry)

[Other nodes](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#other-nodes)

[Open Carol's node](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#open-carol-s-node)

[CORS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#cors)

[Compose elements](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#compose-elements)

[The executables that run](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-executables-that-run)

[The data each container needs](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-data-each-container-needs)

[The networks they run in](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#the-networks-they-run-in)

[Additional settings](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#additional-settings)

[Launching Compose](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#launching-compose)

[Interacting with Compose](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#interacting-with-compose)

[Stopping Compose](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#stopping-compose)

[Self-contained checkers blockchain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#self-contained-checkers-blockchain)

[CosmJS faucet](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#cosmjs-faucet)

[Dockerfile construction, checkers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#dockerfile-construction-checkers)

[Dockerfile construction, faucet](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#dockerfile-construction-faucet)

[Build your standalone images](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#build-your-standalone-images)

[Run your standalone checkers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#run-your-standalone-checkers)

[Test your standalone checkers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html#test-your-standalone-checkers)

#### **Get Cosmos updates**

Unsubscribe at any time. [Privacy Policy↗](https://v1.cosmos.network/privacy)

     Next

Documentation

[Cosmos SDK](https://docs.cosmos.network/)[Cosmos Hub](https://hub.cosmos.network/)[CometBFT](https://docs.cometbft.com/)[IBC Protocol](https://ibc.cosmos.network/)

Community

[Interchain blog](https://blog.cosmos.network/)[Forum](https://forum.cosmos.network/)[Discord](https://discord.gg/cosmosnetwork)

Contributing

[Source code on GitHub](https://github.com/cosmos/sdk-tutorials)

[](https://ida.interchain.io/)

[Interchain Developer Academy](https://ida.interchain.io/)

**[](https://blog.cosmos.network/)[](https://twitter.com/cosmos)[](https://discord.gg/cosmosnetwork)[](https://www.linkedin.com/company/interchain-foundation/about/)[](https://reddit.com/r/cosmosnetwork)[](https://t.me/cosmosproject)[](https://www.youtube.com/c/CosmosProject)**



Dark mode

† This website is maintained by the Interchain Foundation (ICF). The contents and opinions of this website are those of the ICF. The ICF provides links to cryptocurrency exchanges as a service to the public. The ICF does not warrant that the information provided by these websites is correct, complete, and up-to-date. The ICF is not responsible for their content and expressly rejects any liability for damages of any kind resulting from the use, reference to, or reliance on any information contained within these websites.

Cosmos is a registered trademark of the [Interchain Foundation.](https://interchain.io/)[Privacy](https://v1.cosmos.network/privacy)