[](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/4-migration-leaderboard.html)



Search

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

Search



Filters

Interchain Developer Academy

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Docs Version Switcher

On this page

[High level considerations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#high-level-considerations)

[What you will do](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#what-you-will-do)

[v1.1 to v2 leaderboard migration helper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#v1-1-to-v2-leaderboard-migration-helper)

[Types and interfaces](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#types-and-interfaces)

[PlayerInfo extraction](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#playerinfo-extraction)

[Leaderboard computation](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#leaderboard-computation)

[Routines orchestration](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#routines-orchestration)

[Advertising the function to use](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#advertising-the-function-to-use)

[v1.1 to v2 migration proper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#v1-1-to-v2-migration-proper)

[Consensus version and name](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#consensus-version-and-name)

[Callback in leaderboard module](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#callback-in-leaderboard-module)

[Callback in app](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#callback-in-app)

[Unit tests](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#unit-tests)

[New mock types](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#new-mock-types)

[Create the mock instance](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#create-the-mock-instance)

[Configure the mock](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#configure-the-mock)

[Rest of the test](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#rest-of-the-test)

[Run it](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#run-it)

[Interact via the CLI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#interact-via-the-cli)

[Launch v1.1](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#launch-v1-1)

[Add games](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#add-games)

[Governance proposal](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#governance-proposal)

[Launch v2](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#launch-v2)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#migrate-the-leaderboard-module-after-production) **Migrate the Leaderboard Module After Production**



Make sure you have all you need before proceeding:

* You understand the concepts of [Protobuf](https://ida.interchain.io/academy/2-cosmos-concepts/6-protobuf.html), [modules](https://ida.interchain.io/academy/2-cosmos-concepts/5-modules.html), and [migrations](https://ida.interchain.io/academy/2-cosmos-concepts/16-migrations.html).
* Go is installed.
* You have the checkers blockchain codebase up to the *Add a Leaderboard Module*. If not, follow the [previous steps](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/3-add-leaderboard.html) or check out the [relevant version (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/tree/leaderboard-handling).



In this section, you will:

* Add the leaderboard module via migration.
* Populate the module's genesis with a proper leaderboard.

In previous sections:

* You added, and [added a migration for](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html), a player info structure that tallies wins and losses per player. You called it v1.1.
* You added a [leaderboard module](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/3-add-leaderboard.html), which exists only if you start a new blockchain from scratch. You called it v2.

Here you will reuse some learnings from the v1.1 migration, and adjust them for the special case of a new module.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#high-level-considerations) High level considerations

Here the decision is to start your v2's leaderboard as if all played past games had been counted for the leaderboard. You *only* need to go through all player information and add a leaderboard including the information. Migration is a good method to tackle the initial leaderboard.

For the avoidance of doubt, **v1.1 and v2 refer to the overall versions of the application**, and not to the *consensus versions* of individual modules, which may change or not.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#what-you-will-do) What you will do

To prepare for your v1.1-to-v2 migration, you will:

1. Add helper functions to process large amounts of data from the latest chain state of type v1.1.
2. Add a function to migrate your state from v1.1 to v2.
3. Make sure you can handle large amounts of data.
4. Put callbacks if necessary.

*Why do you need to make sure you can handle large amounts of data?* The full state at the point of migration may well have millions of players. You do not want your process to grind to a halt because of a lack of memory or I/O capacity.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#v1-1-to-v2-leaderboard-migration-helper) v1.1 to v2 leaderboard migration helper

In the migration, there are two time-consuming parts:

1. Fetching the stored player info records in a paginated way, consuming mostly database resources.
2. Sorting each intermediate leaderboard, consuming mostly computation resources.

It looks beneficial to use Go routines in this case too, and to use a *player info* channel to pass along arrays of player info records.

In practice, repeatedly building the intermediate leaderboard means adding *k* new Winners to the sorted array, sorting it, clipping it to Params.Length, and repeating. What constitutes a good *k* value should be dictated by testing and performance measurements. However, you can start with your best guess in a new file created for this purpose.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#types-and-interfaces) Types and interfaces

Where do you put this new file? The leaderboard module's consensus version starts at 2. The application will go from *no leaderboard* to *leaderboard at version 2*. So it makes sense to create a new folder to encapsulate this knowledge:



Copy

$ mkdir -p x/leaderboard/migrations/cv2/types

Put your target *k* length in:



Copy

const (

PlayerInfoChunkSize = types.DefaultLength \* uint64(2)

)

x /

leaderboard /

... /

types /

keys.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/migrations/cv2/types/keys.go" \l "L6" \t "_blank)

With a view to reusing the module's Candidate types, you can add a convenience method to convert an array of PlayerInfo to an array of Candidates:



Copy

func MakeCandidatesFromPlayerInfos(playerInfos []checkerstypes.PlayerInfo) (candidates []Candidate, err error) {

candidates = make([]Candidate, 0, len(playerInfos))

for \_, receivedInfo := range playerInfos {

candidate, err := MakeCandidateFromPlayerInfo(receivedInfo)

if err != nil {

return nil, err

}

candidates = append(candidates, candidate)

}

return candidates, nil

}

x /

leaderboard /

... /

types /

leaderboard\_checkers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard//types/leaderboard_checkers.go" \l "L20-L30" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#playerinfo-extraction) PlayerInfo extraction

To extract the PlayerInfo, you need access to a checkers keeper. More precisely, you only need access to its paginated PlayerInfoAll function.

As usual, describe this dependency in expected\_keepers.go:



Copy

type PlayerInfoKeeper interface {

PlayerInfoAll(c context.Context, req \*checkerstypes.QueryAllPlayerInfoRequest) (\*checkerstypes.QueryAllPlayerInfoResponse, error)

}

x /

leaderboard /

types /

expected\_keepers.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/types/expected_keepers.go" \l "L23-L25" \t "_blank)

Then put the migration-specific elements in a dedicated folder:



Copy

$ mkdir x/leaderboard/migrations/cv2/keeper

Create the routine that fetches player info from the checkers storage:



Copy

type PlayerInfosChunk struct {

PlayerInfos []checkerstypes.PlayerInfo

Error error

}

func LoadPlayerInfosToChannel(context context.Context,

playerInfosk types.PlayerInfoKeeper,

playerInfosChannel chan<- PlayerInfosChunk,

chunk uint64) {

defer func() { close(playerInfosChannel) }()

response, err := playerInfosk.PlayerInfoAll(context, &checkerstypes.QueryAllPlayerInfoRequest{

Pagination: &query.PageRequest{Limit: chunk},

})

if err != nil {

playerInfosChannel <- PlayerInfosChunk{PlayerInfos: nil, Error: err}

return

}

playerInfosChannel <- PlayerInfosChunk{PlayerInfos: response.PlayerInfo, Error: nil}

for response.Pagination.NextKey != nil {

response, err = playerInfosk.PlayerInfoAll(context, &checkerstypes.QueryAllPlayerInfoRequest{

Pagination: &query.PageRequest{

Key: response.Pagination.NextKey,

Limit: chunk,

},

})

if err != nil {

playerInfosChannel <- PlayerInfosChunk{PlayerInfos: nil, Error: err}

return

}

playerInfosChannel <- PlayerInfosChunk{PlayerInfos: response.PlayerInfo, Error: nil}

}

}

x /

leaderboard /

... /

keeper /

migration\_leaderboard.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/migrations/cv2/keeper/migration_leaderboard.go" \l "L11-L42" \t "_blank)



Note that:

* This passes along the channel a tuple PlayerInfosChunk that may contain an error. This is to obtain a result similar to when a function returns an optional error.
* It uses the paginated query so as to not overwhelm the memory if there are millions of infos.
* It closes the channel upon exit whether there is an error or not via the use of defer.

This routine populates the player info channel. What about the routine that consumes it?

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#leaderboard-computation) Leaderboard computation

Create the routine function that builds the leaderboard in memory and returns it when complete:



Copy

type LeaderboardResult struct {

Leaderboard \*types.Leaderboard

Error error

}

func HandlePlayerInfosChannel(playerInfosChannel <-chan PlayerInfosChunk,

leaderboardChannel chan<- LeaderboardResult,

leaderboardLength uint64,

addedAt time.Time,

chunk uint64) {

defer func() { close(leaderboardChannel) }()

winners := make([]types.Winner, 0, leaderboardLength+chunk)

for receivedInfos := range playerInfosChannel {

if receivedInfos.Error != nil {

leaderboardChannel <- LeaderboardResult{Leaderboard: nil, Error: receivedInfos.Error}

return

}

if receivedInfos.PlayerInfos != nil {

candidates, err := types.MakeCandidatesFromPlayerInfos(receivedInfos.PlayerInfos)

if err != nil {

leaderboardChannel <- LeaderboardResult{Leaderboard: nil, Error: err}

return

}

winners = types.AddCandidatesAtNow(winners, addedAt, candidates)

if leaderboardLength < uint64(len(winners)) {

winners = winners[:leaderboardLength]

}

}

}

leaderboardChannel <- LeaderboardResult{

Leaderboard: &types.Leaderboard{Winners: winners},

Error: nil,

}

}

x /

leaderboard /

... /

keeper /

migration\_leaderboard.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/migrations/cv2/keeper/migration_leaderboard.go" \l "L44-L77" \t "_blank)



Note that:

* The winners are initialized at a 0 size but with a capacity of Params.Length + chunk, which is the expected maximum intermediate size it will reach. This initialization should ensure that the slice does not need to have its capacity increased mid-process.
* It also passes along a tuple with an optional error.
* It closes the channel it populates upon exit whether there is an error or not via the use of defer.

This routine populates the leaderboard channel. Where is it consumed?

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#routines-orchestration) Routines orchestration

Now you can declare the main function that creates the channels and routines, and collects the leaderboard when done:



Copy

func MapPlayerInfosReduceToLeaderboard(context context.Context,

k keeper.Keeper,

playerInfosk types.PlayerInfoKeeper,

leaderboardLength uint64,

addedAt time.Time,

chunk uint64) (\*types.Leaderboard, error) {

playerInfosChannel := make(chan PlayerInfosChunk)

leaderboardChannel := make(chan LeaderboardResult)

go HandlePlayerInfosChannel(playerInfosChannel, leaderboardChannel, leaderboardLength, addedAt, chunk)

go LoadPlayerInfosToChannel(context, playerInfosk, playerInfosChannel, chunk)

result := <-leaderboardChannel

return result.Leaderboard, result.Error

}

x /

leaderboard /

... /

keeper /

migration\_leaderboard.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/migrations/cv2/keeper/migration_leaderboard.go" \l "L79-L89" \t "_blank)



Note that:

* This returns the leaderboard instead of saving it in the keeper. That is because, when **introducing a module**, you have to initialize it with a **genesis**, and this computed leaderboard will be part of the module's genesis.
* It delegates the closing of channels to the routines.
* It starts the *second* routine first to reduce the likelihood of channel clogging.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#advertising-the-function-to-use) Advertising the function to use

The migration proper needs to introduce the new module and then populate the genesis with the result of the MapPlayerInfosReduceToLeaderboard function. You can encapsulate and advertise this knowledge in two functions:



Copy

func ComputeMigratedLeaderboard(ctx sdk.Context, playerInfosk types.PlayerInfoKeeper) (\*types.Leaderboard, error) {

return cv2keeper.MapPlayerInfosReduceToLeaderboard(

sdk.WrapSDKContext(ctx),

playerInfosk,

types.DefaultLength,

ctx.BlockTime(),

cv2types.PlayerInfoChunkSize)

}

func ComputeInitGenesis(ctx sdk.Context, playerInfosk types.PlayerInfoKeeper) (\*types.GenesisState, error) {

leaderboard, err := ComputeMigratedLeaderboard(ctx, playerInfosk)

if err != nil {

return nil, err

}

return &types.GenesisState{

Params: types.DefaultParams(),

Leaderboard: \*leaderboard,

}, nil

}

x /

leaderboard /

... /

cv2 /

migration.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/migrations/cv2/migration.go" \l "L11-L24" \t "_blank)



To further limit the dependency of the leaderboard module on the checkers module, you could consider:

* Having the new expected interface be based on Candidate and not PlayerInfo.
* And keeping the transformation between one and the other in a file whose name is suffixed with \_checkers.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#v1-1-to-v2-migration-proper) v1.1 to v2 migration proper

You now have in place the functions that will handle the store migration. Next you have to set up the chain of command for these functions to be called by the node at the right point in time.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#consensus-version-and-name) Consensus version and name

The upgrade module keeps in its store the [different module versions (opens new window)↗](https://docs.cosmos.network/main/core/upgrade.html#tracking-module-versions) that are currently running. To signal an upgrade, your module needs to return a different value when queried by the upgrade module. As it stands, your leaderboard consensus version is 2 and that will be its first value when added to the application. To make this explicit, and consistent with the pattern used in the checkers module, you can keep this information in a constant like you did for the checkers module:



Copy

const (

PlayerInfoChunkSize = types.DefaultLength \* uint64(2)

+ ConsensusVersion = uint64(2)

)

x /

leaderboard /

... /

types /

keys.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/migrations/cv2/types/keys.go" \l "L7" \t "_blank)

It can be used by module.go:



Copy

- func (AppModule) ConsensusVersion() uint64 { return 2 }

+ func (AppModule) ConsensusVersion() uint64 { return cv2types.ConsensusVersion }

x /

leaderboard /

module.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/module.go" \l "L172" \t "_blank)



The consensus version number has no connection to v1.1 or v2. The consensus version number is for the module, whereas v1.1 or v2 is for the whole application.

You also have to pick a name for the upgrade you have prepared. This name will identify your specific upgrade when it is mentioned in a Plan (i.e. an upgrade governance proposal). Use a name that is relevant at the application level:



Copy

$ mkdir app/upgrades/v1\_1tov2

In this you save:



Copy

const UpgradeName = "v1\_1tov2"

app /

upgrades /

v1\_1tov2 /

keys.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/app/upgrades/v1_1tov2/keys.go" \l "L3" \t "_blank)

You have to inform your app about:

1. The module being introduced.
2. The genesis to use for it.

Prepare these in turn.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#callback-in-leaderboard-module) Callback in leaderboard module

Although the module does not upgrade per se, you need to make sure that it does not fail when presented with an upgrade. Call RegisterMigration with nil action:



Copy

import (

...

+ cv2Types "github.com/b9lab/checkers/x/checkersleaderboard/migrations/cv2/types"

"github.com/b9lab/checkers/x/leaderboard/types"

...

)

func (am AppModule) RegisterServices(cfg module.Configurator) {

types.RegisterQueryServer(cfg.QueryServer(), am.keeper)

+ if err := cfg.RegisterMigration(types.ModuleName, cv2Types.ConsensusVersion, func(ctx sdk.Context) error {

+ return nil

+ }); err != nil {

+ panic(fmt.Errorf("failed to register cv2 leaderboard migration of %s: %w", types.ModuleName, err))

+ }

}

x /

leaderboard /

module.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/x/leaderboard/module.go" \l "L143-L147" \t "_blank)



You *could* run a proper migration here, if you added a checkers keeper (or rather a types.PlayerInfoKeeper) in your module.   
  
The downside is that this keeper would be used only at the time of migration and therefore is a massive overkill, if not a security risk.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#callback-in-app) Callback in app

In a [previous section](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html) you already prepared app.go to handle a migration, namely v1tov1\_1. Here you add to this setupUpgradeHandlers function in two places:

1. First, after the v1tov1\_1 upgrade method, introduce the way to get the new leaderboard's module genesis:



Copy

// v1 to v1.1 upgrade handler

app.UpgradeKeeper.SetUpgradeHandler(

v1tov1\_1.UpgradeName,

...

)

+ // v1.1 to v2 upgrade handler

+ app.UpgradeKeeper.SetUpgradeHandler(

+ v1\_1tov2.UpgradeName,

+ func(ctx sdk.Context, plan upgradetypes.Plan, vm module.VersionMap) (module.VersionMap, error) {

+ vm[leaderboardmoduletypes.ModuleName] = leaderboardmodulemigrationscv2types.ConsensusVersion

+ genesis, err := leaderboardmodulemigrationscv2.ComputeInitGenesis(ctx, app.CheckersKeeper)

+ if err != nil {

+ return vm, err

+ }

+ gen, err := app.appCodec.MarshalJSON(genesis)

+ if err != nil {

+ return vm, err

+ }

+ app.mm.Modules[leaderboardmoduletypes.ModuleName].InitGenesis(

+ ctx,

+ app.appCodec,

+ gen)

+ return app.mm.RunMigrations(ctx, app.configurator, vm)

+ },

+ )

...

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/app/app.go" \l "L792-L811" \t "_blank)



Note that:

* + The version map is populated with the current consensus version of the leaderboard. This stops the leaderboard from being upgraded further.
  + app.CheckersKeeper is used. This is the only time the leaderboard module has access to the checkers module.
  + The genesis needs to be marshalled before being passed to InitGenesis.

1. Second, inform it that as part of the v1\_1tov2 upgrade a new store key is introduced:



Copy

...

switch upgradeInfo.Name {

case v1tov1\_1.UpgradeName:

+ case v1\_1tov2.UpgradeName:

+ storeUpgrades = &storetypes.StoreUpgrades{

+ Added: []string{leaderboardmoduletypes.StoreKey},

+ }

}

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/leaderboard-migration/app/app.go" \l "L829-L833" \t "_blank)

This is where the genesis will be saved.

With this, the app is configured to handle the module upgrade.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#unit-tests) Unit tests

After all these changes it is worthwhile adding tests, at least on the helpers.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#new-mock-types) New mock types

You introduced a new expected keeper. If you want to unit test your migration helpers properly, you have to mock this new expected interface:

1. Add to the relevant Makefile target:



Copy

mock-expected-keepers:

...

-destination=x/checkers/testutil/expected\_keepers\_mocks.go

+ mockgen -source=x/leaderboard/types/expected\_keepers.go \

+ -package testutil \

+ -destination=x/leaderboard/testutil/expected\_keepers\_mocks.go

1. Now run it:

**Local**

**Docker**



Copy

$ make mock-expected-keepers

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

make mock-expected-keepers

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#create-the-mock-instance) Create the mock instance

In a new file, add a function to prepare your new mock:



Copy

const (

alice = testutil.Alice

bob = testutil.Bob

carol = testutil.Carol

)

func setupMockForLeaderboardMigration(t testing.TB) (context.Context, \*gomock.Controller, \*testutil.MockPlayerInfoKeeper) {

ctrl := gomock.NewController(t)

playerInfoMock := testutil.NewMockPlayerInfoKeeper(ctrl)

\_, ctx := keepertest.LeaderboardKeeper(t)

return sdk.WrapSDKContext(ctx), ctrl, playerInfoMock

}

In your test, prepare the mock:



Copy

func TestComputeLeaderboard(t \*testing.T) {

context, ctrl, playerInfoMock := setupMockForLeaderboardMigration(t)

// TODO

ctrl.Finish()

}

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#configure-the-mock) Configure the mock

To use your mock, this time it takes a bit of effort:

* To test that it was called correctly, use .EXPECT() with defined values.
* To have it return player infos in chunks, use .Return() and make sure the pagination's NextKey is populated when necessary.
* To confirm that one call happens before the other, use gomock.InOrder().

Imagine that the keeper has:



Copy

PlayerInfos: []checkerstypes.PlayerInfo{

{ Index: alice, WonCount: 1 },

{ Index: bob, WonCount: 2 },

{ Index: carol, WonCount: 3 },

}

Also imagine you want to have it return the info paginated in chunks of size 2. This means two calls:

1. The first call:



Copy

context, ctrl, playerInfoMock := setupMockForLeaderboardMigration(t)

- // TODO

+ firstCall := playerInfoMock.EXPECT().

+ PlayerInfoAll(context, &checkerstypes.QueryAllPlayerInfoRequest{

+ Pagination: &query.PageRequest{Limit: 2},

+ }).

+ Return(&checkerstypes.QueryAllPlayerInfoResponse{

+ PlayerInfo: []checkerstypes.PlayerInfo{

+ {

+ Index: alice,

+ WonCount: 1,

+ },

+ {

+ Index: bob,

+ WonCount: 2,

+ },

+ },

+ Pagination: &query.PageResponse{

+ NextKey: []byte("more"),

+ },

+ }, nil)

...

1. The second call:



Copy

firstCall := playerInfoMock.EXPECT().

...

}, nil)

+ secondCall := playerInfoMock.EXPECT().

+ PlayerInfoAll(context, &checkerstypes.QueryAllPlayerInfoRequest{

+ Pagination: &query.PageRequest{

+ Key: []byte("more"),

+ Limit: 2,

+ },

+ }).

+ Return(&checkerstypes.QueryAllPlayerInfoResponse{

+ PlayerInfo: []checkerstypes.PlayerInfo{

+ {

+ Index: carol,

+ WonCount: 3,

+ },

+ },

+ Pagination: &query.PageResponse{

+ NextKey: nil,

+ },

+ }, nil)

...

1. Now specify the desired order:



Copy

secondCall := playerInfoMock.EXPECT().

...

}, nil)

+ gomock.InOrder(firstCall, secondCall)

...

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#rest-of-the-test) Rest of the test

From there, the test is as usual:



Copy

...

gomock.InOrder(firstCall, secondCall)

+ leaderboard, err := cv2keeper.MapPlayerInfosReduceToLeaderboard(

+ context,

+ playerInfoMock,

+ 2,

+ time.Unix(int64(1001), 0),

+ 2)

+ require.Nil(t, err)

+ require.Equal(t, 2, len(leaderboard.Winners))

+ require.EqualValues(t, types.Leaderboard{

+ Winners: []types.Winner{

+ {

+ Address: carol,

+ WonCount: 3,

+ AddedAt: 1001,

+ },

+ {

+ Address: bob,

+ WonCount: 2,

+ AddedAt: 1001,

+ },

+ },

+ }, \*leaderboard)

ctrl.Finish()

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#run-it) Run it

You can confirm that the test passes by running:

**Local**

**Docker**



Copy

$ go test github.com/b9lab/checkers/x/leaderboard/migrations/cv2/keeper

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

go test github.com/b9lab/checkers/x/leaderboard/migrations/cv2/keeper

Given the configuration difficulty of the mock, only this test will do.

It is not possible to add integration tests on the migration proper, because when the app is created it is already at v2.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#interact-via-the-cli) Interact via the CLI

You can execute a live upgrade from the command line. The following upgrade process takes inspiration from [this one (opens new window)↗](https://hub.cosmos.network/main/hub-tutorials/live-upgrade-tutorial.html) based on Gaia. You will:

* Check out the checkers v1.1 code.
* Build the v1.1 checkers executable.
* Initialize a local blockchain and network.
* Run v1.1 checkers.
* Add one or more incomplete games.
* Add one or more complete games with the help of a CosmJS integration test.
* Create a governance proposal to upgrade with the right plan name at an appropriate block height.
* Make the proposal pass.
* Wait for v1.1 checkers to halt on its own at the upgrade height.
* Check out the checkers v2 code.
* Build the v2 checkers executable.
* Run v2 checkers.
* Confirm that you now have a correct leaderboard.

Start your engines!

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#launch-v1-1) Launch v1.1

After committing your changes, in a shell checkout v1.1 of checkers with the content of the CosmJS client work:



Copy

$ git checkout player-info-migration

$ git submodule update --init

Build the v1.1 executable for your platform:

**Local**

**Docker**



Copy

$ go build -o release/v1\_1/checkersd cmd/checkersd/main.go

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

go build -o release/v1\_1/checkersd cmd/checkersd/main.go

With the release/v1\_1/checkersd executable ready, you can initialize the network.



Because this is an exercise, to avoid messing with your keyring you must always specify --keyring-backend test.

Add two players:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd keys add alice --keyring-backend test

$ ./release/v1\_1/checkersd keys add bob --keyring-backend test

Copy

$ docker network create checkers-net

$ docker create -it \

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

--network checkers-net \

--name checkers \

-p 26657:26657 \

checkers\_i

$ docker start checkers

$ docker exec -t checkers \

./release/v1\_1/checkersd keys add alice --keyring-backend test

$ docker exec -t checkers \

./release/v1\_1/checkersd keys add bob --keyring-backend test

You should not use docker run --rm here, because when checkersd stops you do not want to remove the container and thereby destroy the saved keys, and the future genesis too. Instead, you reuse them all in the next calls.

Create a new genesis:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd init checkers --chain-id checkers-1

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd init checkers --chain-id checkers-1

Give your players the same token amounts that were added by Ignite, as found in config.yml:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd add-genesis-account \

alice 200000000stake,20000token --keyring-backend test

$ ./release/v1\_1/checkersd add-genesis-account \

bob 100000000stake,10000token --keyring-backend test

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd add-genesis-account \

alice 200000000stake,20000token --keyring-backend test

$ docker exec -t checkers \

./release/v1\_1/checkersd add-genesis-account \

bob 100000000stake,10000token --keyring-backend test

To be able to run a quick test, you need to change the voting period of a proposal. This is found in the genesis:

**Local**

**Docker**



Copy

$ jq '.app\_state.gov.voting\_params.voting\_period' ~/.checkers/config/genesis.json

Copy

$ docker exec -it checkers \

jq '.app\_state.gov.voting\_params.voting\_period' /root/.checkers/config/genesis.json

This returns something like:



Copy

"172800s"

That is two days, which is too long to wait for CLI tests. Choose another value, perhaps 10 minutes (i.e. "600s"). Update it in place in the genesis:

**Local**

**Docker**



Copy

$ cat <<< $(jq '.app\_state.gov.voting\_params.voting\_period = "600s"' ~/.checkers/config/genesis.json) \

> ~/.checkers/config/genesis.json

Copy

$ docker exec -t checkers \

bash -c "cat <<< \$(jq '.app\_state.gov.voting\_params.voting\_period = \"600s\"' /root/.checkers/config/genesis.json) \

> /root/.checkers/config/genesis.json"

You can confirm that the value is in using the earlier command.

Make Alice the chain's validator too by creating a genesis transaction modeled on that done by Ignite, as found in config.yml:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd gentx alice 100000000stake \

--keyring-backend test --chain-id checkers-1

$ ./release/v1\_1/checkersd collect-gentxs

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd gentx alice 100000000stake \

--keyring-backend test --chain-id checkers-1

$ docker exec -t checkers \

./release/v1\_1/checkersd collect-gentxs

Now you can start the chain proper:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd start

Copy

$ docker exec -it checkers \

./release/v1\_1/checkersd start \

--rpc.laddr "tcp://0.0.0.0:26657"

Note that you need to force the node to listen on all IP addresses, not just 127.0.0.1 as it would do by default.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#add-games) Add games

From another shell, create a few un-played games with:

**Local**

**Docker**



Copy

$ export alice=$(./release/v1\_1/checkersd keys show alice -a --keyring-backend test)

$ export bob=$(./release/v1\_1/checkersd keys show bob -a --keyring-backend test)

$ ./release/v1\_1/checkersd tx checkers create-game \

$alice $bob 10 stake \

--from $alice --keyring-backend test --yes \

--chain-id checkers-1 \

--broadcast-mode block

Copy

$ export alice=$(docker exec checkers ./release/v1\_1/checkersd keys show alice -a --keyring-backend test)

$ export bob=$(docker exec checkers ./release/v1\_1/checkersd keys show bob -a --keyring-backend test)

$ docker exec -t checkers \

./release/v1\_1/checkersd tx checkers create-game \

$alice $bob 10 stake \

--from $alice --keyring-backend test --yes \

--chain-id checkers-1 \

--broadcast-mode block



The --broadcast-mode block flag means that you can fire up many such games by just copying the command without facing any sequence errors.

To get a few complete games, you are going to run the [integration tests (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/main/test/integration/stored-game-action.ts) against it. These tests call a faucet if the accounts do not have enough. Because you do not have a faucet here, you need to credit your test accounts with standard bank send transactions. You can use the same values as found in the before:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd tx bank \

send $alice cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql 300stake \

--from $alice --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

$ ./release/v1\_1/checkersd tx bank \

send $alice cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql 10token \

--from $alice --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

$ ./release/v1\_1/checkersd tx bank \

send $bob cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22 300stake \

--from $bob --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

$ ./release/v1\_1/checkersd tx bank \

send $bob cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22 10token \

--from $bob --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd tx bank \

send $alice cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql 300stake \

--from $alice --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

$ docker exec -t checkers \

./release/v1\_1/checkersd tx bank \

send $alice cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql 10token \

--from $alice --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

$ docker exec -t checkers \

./release/v1\_1/checkersd tx bank \

send $bob cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22 300stake \

--from $bob --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

$ docker exec -t checkers \

./release/v1\_1/checkersd tx bank \

send $bob cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22 10token \

--from $bob --keyring-backend test \

--chain-id checkers-1 \

--broadcast-mode block --yes

With the test accounts sufficiently credited, you can now run the integration tests. Run them three times in a row to create three complete games:

**Local**

**Docker**



Copy

$ pushd client && npm test && popd

Copy

$ docker run --rm -it \

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

-w /client \

--network checkers-net \

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

node:18.7-slim \

npm test

You can confirm that you have computed player info:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd query checkers \

list-player-info --output json \

| jq '.playerInfo'

Copy

$ docker exec -t checkers \

bash -c "./release/v1\_1/checkersd query checkers \

list-player-info --output json \

| jq '.playerInfo'"

With enough games in the system, you can move to the software upgrade governance proposal.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#governance-proposal) Governance proposal

For the software upgrade governance proposal, you want to make sure that it stops the chain not too far in the future but still after the voting period. With a voting period of 10 minutes, take 15 minutes. How many seconds does a block take?

**Local**

**Docker**



Copy

$ jq -r ".app\_state.mint.params.blocks\_per\_year" \

~/.checkers/config/genesis.json

Copy

$ docker exec -t checkers \

bash -c 'jq -r ".app\_state.mint.params.blocks\_per\_year" /root/.checkers/config/genesis.json'

This returns something like:



Copy

6311520

That many blocks\_per\_year computes down to 5 seconds per block. At this rate, 15 minutes mean 180 blocks.

What is the current block height? Check:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd status \

| jq -r ".SyncInfo.latest\_block\_height"

Copy

$ docker exec -t checkers \

bash -c './release/v1\_1/checkersd status \

| jq -r ".SyncInfo.latest\_block\_height"'

This returns something like:



Copy

1000

That means you will use:



Copy

--upgrade-height 1180

What is the minimum deposit for a proposal? Check:

**Local**

**Docker**



Copy

$ jq ".app\_state.gov.deposit\_params.min\_deposit" \

~/.checkers/config/genesis.json

Copy

$ docker exec -t checkers \

bash -c 'jq ".app\_state.gov.deposit\_params.min\_deposit" \

/root/.checkers/config/genesis.json'

This returns something like:



Copy

[

{

"denom": "stake",

"amount": "10000000"

}

]

This is the minimum amount that Alice has to deposit when submitting the proposal. This will do:



Copy

--deposit 10000000stake

Now submit your governance proposal upgrade:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd tx gov submit-proposal software-upgrade v1\_1tov2 \

--title "v1\_1tov2" \

--description "Increase engagement via the use of a leaderboard" \

--from $alice --keyring-backend test --yes \

--chain-id checkers-1 \

--broadcast-mode block \

--upgrade-height 1180 \

--deposit 10000000stake

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd tx gov submit-proposal software-upgrade v1\_1tov2 \

--title "v1\_1tov2" \

--description "Increase engagement via the use of a leaderboard" \

--from $alice --keyring-backend test --yes \

--chain-id checkers-1 \

--broadcast-mode block \

--upgrade-height 1180 \

--deposit 10000000stake

This returns something like:



Copy

...

type: proposal\_deposit

- attributes:

- key: proposal\_id

value: "1"

- key: proposal\_type

value: SoftwareUpgrade

- key: voting\_period\_start

value: "1"

...

Where 1 is the proposal ID you reuse. Have Alice and Bob vote yes on it:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd tx gov vote 1 yes \

--from $alice --keyring-backend test --yes \

--chain-id checkers-1

$ ./release/v1\_1/checkersd tx gov vote 1 yes \

--from $bob --keyring-backend test --yes \

--chain-id checkers-1

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd tx gov vote 1 yes \

--from $alice --keyring-backend test --yes \

--chain-id checkers-1

$ docker exec -t checkers \

./release/v1\_1/checkersd tx gov vote 1 yes \

--from $bob --keyring-backend test --yes \

--chain-id checkers-1

Confirm that it has collected the votes:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd query gov votes 1

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd query gov votes 1

It should print:



Copy

votes:

- option: VOTE\_OPTION\_YES

options:

- option: VOTE\_OPTION\_YES

weight: "1.000000000000000000"

proposal\_id: "1"

voter: cosmos1hzftnstmlzqfaj0rz39hn5pe2vppz0phy4x9ct

- option: VOTE\_OPTION\_YES

options:

- option: VOTE\_OPTION\_YES

weight: "1.000000000000000000"

proposal\_id: "1"

voter: cosmos1hj2x82j49fv90tgtdxrdw5fz3w2vqeqqjhrxle

See how long you have to wait for the chain to reach the end of the voting period:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd query gov proposal 1

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd query gov proposal 1

In the end this prints:



Copy

...

status: PROPOSAL\_STATUS\_VOTING\_PERIOD

...

voting\_end\_time: "2022-08-25T10:38:22.240766103Z"

...

Wait for this period. Afterward, with the same command you should see:



Copy

...

status: PROPOSAL\_STATUS\_PASSED

...

Now wait for the chain to reach the desired block height, which should take five more minutes as per your parameters. When it has reached that height, the shell with the running checkersd should show something like:



Copy

...

6:29PM INF finalizing commit of block hash=E6CB6F1E8CF4699543950F756F3E15AE447701ABAC498CDBA86633AC93A73EE7 height=1180 module=consensus num\_txs=0 root=21E51E52AA3F06BE59C78CE11D3171E6F7240D297E4BCEAB07FC5A87957B3BE2

6:29PM ERR UPGRADE "v1\_1tov2" NEEDED at height: 1180:

6:29PM ERR CONSENSUS FAILURE!!! err="UPGRADE \"v1\_1tov2\" NEEDED at height: 1180: " module=consensus stack="goroutine 62 [running]:\nruntime/debug.Stack

...

6:29PM INF Stopping baseWAL service impl={"Logger":{}} module=consensus wal=/root/.checkers/data/cs.wal/wal

6:29PM INF Stopping Group service impl={"Dir":"/root/.checkers/data/cs.wal","Head":{"ID":"ZsAlN7DEZAbV:/root/.checkers/data/cs.wal/wal","Path":"/root/.checkers/data/cs.wal/wal"},"ID":"group:ZsAlN7DEZAbV:/root/.checkers/data/cs.wal/wal","Logger":{}} module=consensus wal=/root/.checkers/data/cs.wal/wal

...

At this point, run in another shell:

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd status \

| jq -r ".SyncInfo.latest\_block\_height"

Copy

$ docker exec -it checkers \

bash -c './release/v1\_1/checkersd status \

| jq -r ".SyncInfo.latest\_block\_height"'

You should always get the same value, no matter how many times you try. That is because the chain has stopped. For instance:



Copy

1180

Stop checkersd with CTRL-C. It has saved a new file:

**Local**

**Docker**



Copy

$ cat ~/.checkers/data/upgrade-info.json

Copy

$ docker exec -it checkers \

cat /root/.checkers/data/upgrade-info.json

This prints:



Copy

{"name":"v1\_1tov2","height":1180}

With your node (and therefore your whole blockchain) down, you are ready to move to v2.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#launch-v2) Launch v2

With v1\_1 stopped and its state saved, it is time to move to v2. Checkout v2 of checkers:



Copy

$ git checkout leaderboard-migration

Back in the first shell, build the v2 executable:

**Local**

**Docker**



Copy

$ go build -o ./release/v2/checkersd ./cmd/checkersd/main.go

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

go build -o ./release/v2/checkersd ./cmd/checkersd/main.go

Launch it:

**Local**

**Docker**



Copy

$ ./release/v2/checkersd start

Copy

$ docker exec -it checkers \

./release/v2/checkersd start \

--rpc.laddr "tcp://0.0.0.0:26657"

It should start and display something like:



Copy

...

7:06PM INF applying upgrade "v1\_1tov2" at height: 1180

7:06PM INF Leaderboard genesis saved

...

After it has started, you can confirm in another shell that you have the expected leaderboard with:

**Local**

**Docker**



Copy

$ ./release/v2/checkersd query leaderboard show-leaderboard

Copy

$ docker exec -t checkers \

./release/v2/checkersd query leaderboard show-leaderboard

This should print something like:



Copy

Leaderboard:

winners:

- addedAt: "1682983659"

address: cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql

wonCount: "3"

Note how it took the time of the block when v1\_1 stopped.

Congratulations, you have upgraded your blockchain almost as if in production!

You can stop Ignite CLI. If you used Docker that would be:



Copy

$ docker stop checkers

$ docker rm checkers

$ docker network rm checkers-net

Your checkers blockchain is done! It has a leaderboard, which was introduced later in production thanks to migrations.

You no doubt have many ideas about how to improve it. In particular, you could implement the missing *draw* mechanism, which in effect has to be accepted by both players.

synopsis

To summarize, this section has explored:

* How to add a leaderboard to an existing blockchain.
* How to upgrade a blockchain in production, by migrating from v1\_1 of the blockchain to v2, and the new store structure that will be introduced by the upgrade.
* How to handle the data migrations and logic upgrades implicit during migration, such as with the use of helper functions.
* Worthwhile unit tests with regard to leaderboard handling.
* A complete procedure for how to conduct the update via the CLI.

previous

[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/3-add-leaderboard.html)

**[Add a Leaderboard as a Module](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/3-add-leaderboard.html)**

up next

**[Simulate a Migration in Docker](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/5-migration-prod.html)**

[[](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/5-migration-prod.html)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/5-migration-prod.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

[High level considerations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#high-level-considerations)

[What you will do](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#what-you-will-do)

[v1.1 to v2 leaderboard migration helper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#v1-1-to-v2-leaderboard-migration-helper)

[Types and interfaces](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#types-and-interfaces)

[PlayerInfo extraction](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#playerinfo-extraction)

[Leaderboard computation](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#leaderboard-computation)

[Routines orchestration](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#routines-orchestration)

[Advertising the function to use](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#advertising-the-function-to-use)

[v1.1 to v2 migration proper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#v1-1-to-v2-migration-proper)

[Consensus version and name](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#consensus-version-and-name)

[Callback in leaderboard module](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#callback-in-leaderboard-module)

[Callback in app](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#callback-in-app)

[Unit tests](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#unit-tests)

[New mock types](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#new-mock-types)

[Create the mock instance](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#create-the-mock-instance)

[Configure the mock](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#configure-the-mock)

[Rest of the test](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#rest-of-the-test)

[Run it](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#run-it)

[Interact via the CLI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#interact-via-the-cli)

[Launch v1.1](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#launch-v1-1)

[Add games](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#add-games)

[Governance proposal](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#governance-proposal)

[Launch v2](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/4-migration-leaderboard.html#launch-v2)

#### **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)