[](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/2-migration-info.html)



Search

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

Search



Filters

Interchain Developer Academy

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

[Week 0 - Getting Started](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

[Blockchain 101](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

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

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

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

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

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

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

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

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

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

[Docker Introduction](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)

[Week 1 - Introduction to the Interchain](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[Week 1 Quiz](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)

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

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

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

[Run a Node, API, and CLI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

[Create and Save a Game Properly](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

[Add a Way to Make a Move](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

[Week 2 Exercise](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)

[Week 3 - Introduction to IBC and CosmJS](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

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

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

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

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

[Create a Custom IBC Middleware (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

[Integrate IBC Middleware Into a Chain (OPTIONAL)](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

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

[Create Custom CosmJS Interfaces](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)

[Week 4 - Ignite CLI and IBC Advanced](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

[Keep an Up-To-Date Game Deadline](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

[Keep Track Of How Many Moves Have Been Played](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

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

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

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

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

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

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

[Hermes Relayer](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)

[Week 5 - CosmJS Advanced](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

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

[Backend Script for Game Indexing](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)

[Week 6 - IBC Deep Dive](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

[Extend the Checkers Game With a Leaderboard](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

[Create a Leaderboard Chain](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)

[Week 7 - From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

[From Code to MVP to Production and Migrations](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

[Prepare and Connect to Other Nodes](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

[Configure, Run, and Set Up a Service](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

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

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

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

[Migrate the Leaderboard Module After Production](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html)

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

[Final Exam](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)

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

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

Docs Version Switcher

On this page

[Introducing a new data structure](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#introducing-a-new-data-structure)

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

[New v1.1 information](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#new-v1-1-information)

[v1.1 player information helpers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-1-player-information-helpers)

[v1.1 player information handling](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-1-player-information-handling)

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

[Player info handling unit tests](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#player-info-handling-unit-tests)

[v1 to v1.1 player information migration helper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-to-v1-1-player-information-migration-helper)

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

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

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

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

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

[v1 to v1.1 migration proper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-to-v1-1-migration-proper)

[Consensus version and name](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#consensus-version-and-name)

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

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

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

[Interact via the CLI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#interact-via-the-cli)

[Launch v1](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#launch-v1)

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

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

[Launch v1.1](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#launch-v1-1)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#tally-player-info-after-production) **Tally Player Info 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) 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 *Simulate Production in Docker*. If not, follow the [previous steps](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/1-run-prod-docker.html) or check out the [relevant version (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/tree/run-prod).



In this section, you will:

* Add a new storage structure to tally player information.
* Upgrade your blockchain in production.
* Deal with data migrations and logic upgrades.

If you have been running *v1* of your checkers blockchain for a while, games have been created, played on, won, and lost. In this section, you will introduce *v1.1* of your blockchain where wins and losses and tallied in a new storage data structure.



This is not done in vain. Instead, looking forward, this is done to support the addition of a leaderboard module for your *v2* in the [next section](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/3-add-leaderboard.html).

For now, a good tally should be such that for any player who has **ever** played it should be possible to access a tally of games won. While you are at it, you will add games lost and forfeited. Fortunately, this is possible because all past games and their outcomes are kept in the chain's state. Migration is a good method to tackle the initial tally.

For the avoidance of doubt, **v1 and v1.1 refer to the overall versions of the application**, and not to the *consensus versions* of individual modules, which may change or not. As it happens, your application has a single module, apart from those coming from the Cosmos SDK.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#introducing-a-new-data-structure) Introducing a new data structure

Several things need to be addressed before you can focus all your attention on the migration:

1. Save and mark as v1 the current data types about to be modified with the new version. Data types that will remain unmodified need not be identified as such.
2. Prepare your v1.1 blockchain:
   1. Define your new data types.
   2. Add helper functions to encapsulate clearly defined actions.
   3. Adjust the existing code to make use of and update the new data types.
3. Prepare for your v1-to-v1.1 migration:
   1. Add helper functions to process large amounts of data from the latest chain state of v1.
   2. Add a function to migrate your state from v1 to v1.1.
   3. Make sure you can handle large amounts of data.

*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 games. 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/2-migration-info.html#preparation) Preparation

For your convenience, you decide to keep all the migration steps in a new folder, x/checkers/migrations, and subfolders, which needs to be created:



Copy

$ mkdir x/checkers/migrations

Your data types are defined at a given consensus version of the module, not the application level v1. Find out the checkers module's current consensus version:



Copy

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

x /

checkers /

module.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/x/checkers/module.go" \l "L166" \t "_blank)

Keep a note of it. At some point, you will create a cv2 subfolder (Where cv is short for *consensus version*) for anything related to the consensus version at this level.

If your migration happened to require the *old* data structure at an earlier consensus version, you would save the old types here.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#new-v1-1-information) New v1.1 information

It is time to take a closer look at the new data structures being introduced with the version upgrade.



If you feel unsure about creating new data structures with Ignite CLI, look at the [previous sections](https://ida.interchain.io/hands-on-exercise/1-ignite-cli/4-create-message.html) of the exercise again.

To give the new v1.1 information a data structure, you need the following:

1. Add a set of **stats per player**: it makes sense to save one struct for each player and to map it by address. Remember that a game is stored at a *notional* [StoredGame/value/123/ (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/x/checkers/keeper/stored_game.go#L24-L28), where [StoredGame/value/ (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/run-prod/x/checkers/types/key_stored_game.go#L9) is a constant prefix. Similarly, Ignite CLI creates a new constant to use as the prefix for players:

**Local**

**Docker**



Copy

$ ignite scaffold map playerInfo \

wonCount:uint lostCount:uint forfeitedCount:uint \

--module checkers --no-message

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

ignite scaffold map playerInfo \

wonCount:uint lostCount:uint forfeitedCount:uint \

--module checkers --no-message



The new PlayerInfo/value/ prefix for players helps differentiate between the value for players and the value for games prefixed with StoredGame/value/.   
  
Now you can safely have both StoredGame/value/123/ and PlayerInfo/value/123/ side by side in storage.

This creates a Protobuf file:



Copy

message PlayerInfo {

string index = 1;

uint64 wonCount = 2;

uint64 lostCount = 3;

uint64 forfeitedCount = 4;

}

proto /

checkers /

player\_info.proto

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-object/proto/checkers/player_info.proto" \l "L6-L11" \t "_blank)

It also added the map of new objects to the genesis, effectively your v1.1 genesis:



Copy

import "checkers/player\_info.proto";

message GenesisState {

...

+ repeated PlayerInfo playerInfoList = 4 [(gogoproto.nullable) = false];

}

proto /

checkers /

genesis.proto

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-object/proto/checkers/genesis.proto" \l "L18" \t "_blank)

You will use the player's address as a key to the map.

With the structure set up, it is time to add the code using these new elements in normal operations, before thinking about any migration.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-1-player-information-helpers) v1.1 player information helpers

When a game reaches its resolution, one of the counts needs to add +1.

****

**A detailed look into the code**

To start, add a private helper function that gets the stats from the storage, updates the numbers as instructed, and saves it back:

Copy

func mustAddDeltaGameResultToPlayer(

k \*Keeper,

ctx sdk.Context,

player sdk.AccAddress,

wonDelta uint64,

lostDelta uint64,

forfeitedDelta uint64,

) (playerInfo types.PlayerInfo) {

playerInfo, found := k.GetPlayerInfo(ctx, player.String())

if !found {

playerInfo = types.PlayerInfo{

Index: player.String(),

WonCount: 0,

LostCount: 0,

ForfeitedCount: 0,

}

}

playerInfo.WonCount += wonDelta

playerInfo.LostCount += lostDelta

playerInfo.ForfeitedCount += forfeitedDelta

k.SetPlayerInfo(ctx, playerInfo)

return playerInfo

}

x

checkers

keeper

player\_info\_handler.go

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/player_info_handler.go" \l "L11-L33" \t "_blank)

You can easily call this from these public one-liner functions added to the keeper:

Copy

func (k \*Keeper) MustAddWonGameResultToPlayer(ctx sdk.Context, player sdk.AccAddress) types.PlayerInfo {

return mustAddDeltaGameResultToPlayer(k, ctx, player, 1, 0, 0)

}

func (k \*Keeper) MustAddLostGameResultToPlayer(ctx sdk.Context, player sdk.AccAddress) types.PlayerInfo {

return mustAddDeltaGameResultToPlayer(k, ctx, player, 0, 1, 0)

}

func (k \*Keeper) MustAddForfeitedGameResultToPlayer(ctx sdk.Context, player sdk.AccAddress) types.PlayerInfo {

return mustAddDeltaGameResultToPlayer(k, ctx, player, 0, 0, 1)

}

x

checkers

keeper

player\_info\_handler.go

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/player_info_handler.go" \l "L35-L45" \t "_blank)

Which player should get +1, and on what count? You need to identify the loser and the winner of a game to determine this. Create another private helper:

Copy

func getWinnerAndLoserAddresses(storedGame \*types.StoredGame) (winnerAddress sdk.AccAddress, loserAddress sdk.AccAddress) {

if storedGame.Winner == rules.PieceStrings[rules.NO\_PLAYER] {

panic(types.ErrThereIsNoWinner.Error())

}

redAddress, err := storedGame.GetRedAddress()

if err != nil {

panic(err.Error())

}

blackAddress, err := storedGame.GetBlackAddress()

if err != nil {

panic(err.Error())

}

if storedGame.Winner == rules.PieceStrings[rules.RED\_PLAYER] {

winnerAddress = redAddress

loserAddress = blackAddress

} else if storedGame.Winner == rules.PieceStrings[rules.BLACK\_PLAYER] {

winnerAddress = blackAddress

loserAddress = redAddress

} else {

panic(fmt.Sprintf(types.ErrWinnerNotParseable.Error(), storedGame.Winner))

}

return winnerAddress, loserAddress

}

x

checkers

keeper

player\_info\_handler.go

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/player_info_handler.go" \l "L47-L69" \t "_blank)

You can call this from these public helper functions added to the keeper:

Copy

func (k \*Keeper) MustRegisterPlayerWin(ctx sdk.Context, storedGame \*types.StoredGame) (winnerInfo types.PlayerInfo, loserInfo types.PlayerInfo) {

winnerAddress, loserAddress := getWinnerAndLoserAddresses(storedGame)

return k.MustAddWonGameResultToPlayer(ctx, winnerAddress),

k.MustAddLostGameResultToPlayer(ctx, loserAddress)

}

func (k \*Keeper) MustRegisterPlayerForfeit(ctx sdk.Context, storedGame \*types.StoredGame) (winnerInfo types.PlayerInfo, forfeiterInfo types.PlayerInfo) {

winnerAddress, loserAddress := getWinnerAndLoserAddresses(storedGame)

return k.MustAddWonGameResultToPlayer(ctx, winnerAddress),

k.MustAddForfeitedGameResultToPlayer(ctx, loserAddress)

}

x

checkers

keeper

player\_info\_handler.go

[View source](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/player_info_handler.go" \l "L71-L81" \t "_blank)

Be aware of the two new error types: [ErrThereIsNoWinner (opens new window)](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/types/errors.go#L28) and [ErrWinnerNotParseable (opens new window)](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/types/errors.go#L27).

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-1-player-information-handling) v1.1 player information handling

Now call your helper functions:

1. On a win:



Copy

...

if storedGame.Winner == rules.PieceStrings[rules.NO\_PLAYER] {

...

} else {

...

k.Keeper.MustPayWinnings(ctx, &storedGame)

+ k.Keeper.MustRegisterPlayerWin(ctx, &storedGame)

}

...

x /

checkers /

keeper /

msg\_server\_play\_move.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/msg_server_play_move.go" \l "L80" \t "_blank)

1. On a forfeit:



Copy

...

if storedGame.MoveCount <= 1 {

...

} else {

...

k.MustPayWinnings(ctx, &storedGame)

+ k.MustRegisterPlayerForfeit(ctx, &storedGame)

}

...

x /

checkers /

keeper /

end\_block\_server\_game.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/end_block_server_game.go" \l "L57" \t "_blank)

Your player info tallies are now updated and saved on an on-going basis in your running v1.1 blockchain.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#unit-tests) Unit tests

With all these changes, it is worthwhile adding tests.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#player-info-handling-unit-tests) Player info handling unit tests

Confirm with new tests that the player's information is created or updated on a win, a loss, and a forfeit. For instance, after a winning move:



Copy

func TestCompleteGameAddPlayerInfo(t \*testing.T) {

msgServer, keeper, context, ctrl, escrow := setupMsgServerWithOneGameForPlayMove(t)

ctx := sdk.UnwrapSDKContext(context)

defer ctrl.Finish()

escrow.ExpectAny(context)

testutil.PlayAllMoves(t, msgServer, context, "1", bob, carol, testutil.Game1Moves)

bobInfo, found := keeper.GetPlayerInfo(ctx, bob)

require.True(t, found)

require.EqualValues(t, types.PlayerInfo{

Index: bob,

WonCount: 1,

LostCount: 0,

ForfeitedCount: 0,

}, bobInfo)

carolInfo, found := keeper.GetPlayerInfo(ctx, carol)

require.True(t, found)

require.EqualValues(t, types.PlayerInfo{

Index: carol,

WonCount: 0,

LostCount: 1,

ForfeitedCount: 0,

}, carolInfo)

}

func TestCompleteGameUpdatePlayerInfo(t \*testing.T) {

msgServer, keeper, context, ctrl, escrow := setupMsgServerWithOneGameForPlayMove(t)

ctx := sdk.UnwrapSDKContext(context)

defer ctrl.Finish()

escrow.ExpectAny(context)

keeper.SetPlayerInfo(ctx, types.PlayerInfo{

Index: bob,

WonCount: 1,

LostCount: 2,

ForfeitedCount: 3,

})

keeper.SetPlayerInfo(ctx, types.PlayerInfo{

Index: carol,

WonCount: 4,

LostCount: 5,

ForfeitedCount: 6,

})

testutil.PlayAllMoves(t, msgServer, context, "1", bob, carol, testutil.Game1Moves)

bobInfo, found := keeper.GetPlayerInfo(ctx, bob)

require.True(t, found)

require.EqualValues(t, types.PlayerInfo{

Index: bob,

WonCount: 2,

LostCount: 2,

ForfeitedCount: 3,

}, bobInfo)

carolInfo, found := keeper.GetPlayerInfo(ctx, carol)

require.True(t, found)

require.EqualValues(t, types.PlayerInfo{

Index: carol,

WonCount: 4,

LostCount: 6,

ForfeitedCount: 6,

}, carolInfo)

}

Expand



x /

checkers /

keeper /

msg\_server\_play\_move\_winner\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/msg_server_play_move_winner_test.go" \l "L68-L131" \t "_blank)

You can add similar tests that confirm that nothing happens after a [game creation (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/msg_server_create_game_test.go#L414-L493) or a [non-winning move (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/msg_server_play_move_test.go#L538-L596). You should also check that a [forfeit is registered (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/end_block_server_game_test.go#L577-L682).

This completes your checkers v1.1 chain. If you were to start it anew as is, it would work. However, you already have the v1 of checkers running, so you need to migrate everything.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-to-v1-1-player-information-migration-helper) v1 to v1.1 player information migration helper

With your v1.1 blockchain now fully operational on its own, it is time to work on the issue of stored data migration.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#consensus-version) Consensus version

Your checkers module's current consensus version is 2. You are about to migrate its store, so you need to increment the module's consensus version by 1 exactly (to avoid any future surprises). You should make these numbers explicit:

1. Save the v1 consensus version in a new file:



Copy

const (

ConsensusVersion = uint64(2)

)

x /

checkers /

... /

types /

keys.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migration/cv2/types/keys.go" \l "L4" \t "_blank)

1. Similarly, save the new v1.1 consensus version in another new file:



Copy

const (

ConsensusVersion = uint64(3)

)

x /

checkers /

... /

types /

keys.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migration/cv3/types/keys.go" \l "L4" \t "_blank)

1. Inform the module that it is now at the new consensus version:



Copy

import (

...

+ cv3Types "github.com/b9lab/checkers/x/checkers/migrations/cv3/types"

)

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

+ func (AppModule) ConsensusVersion() uint64 { return cv3Types.ConsensusVersion }

x /

checkers /

module.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/module.go" \l "L167" \t "_blank)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#problem-description) Problem description

Coming back to the store migration, in other words, you need to tackle the creation of player information. You will build the player information by extracting it from all the existing stored games. In the [map/reduce (opens new window)↗](https://en.wikipedia.org/wiki/MapReduce) parlance, you will *reduce* this information from the stored games.

If performance and hardware constraints were not an issue, an easy way to do it would be the following:

1. Call [keeper.GetAllStoredGame() (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/stored_game.go#L50) to get an array with all the games.
2. Keep only the games that have a winner.
3. Then for each game:
   1. Call keeper.GetPlayerInfo or, if that is not found, create player info both for the black player and the red player.
   2. Do +1 on .WonCount or .LostCount according to the game.Winner field. In the current saved state, there is no way to differentiate between a *normal* win and a win by forfeit.
   3. Call keeper.SetPlayerInfo for both black and red players.

Of course, given inevitable resource limitations, you would run into the following problems:

1. Getting all the games in a single array may not be possible, because your node's RAM may not be able to keep a million of them in memory. Or maybe it fails at 100,000 of them.
2. Calling .GetPlayerInfo and .SetPlayerInfo twice per game just to do +1 adds up quickly. Remember that both of these calls are database calls. You could be facing a 12-hour job, during which your chain is offline.
3. Doing it all in a sequential manner would take even more time, as each blocking call blocks the whole process.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#proposed-solution) Proposed solution

Fortunately, there exist ways to mitigate these limitations:

1. You do not need to get all the games at once. The [keeper.StoredGameAll (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-handling/x/checkers/keeper/grpc_query_stored_game.go#L14) function offers pagination. With this, you can limit the impact on the RAM requirement, at the expense of multiple queries.
2. Within each subset of games, you can compute in memory the player list and how many wins and losses each player has. With this *mapping* done, you can add the (in-memory) intermediary WonCount and LostCount sums to each player's stored sums. With this, a +1 is potentially replaced by a +k, at once reducing the number of calls to .GetPlayerInfo and .SetPlayerInfo.
3. You can separate the different calls and computations into [Go routines (opens new window)↗](https://gobyexample.com/goroutines) so that a blocking call does not prevent other computations from taking place in the meantime.

The routines will use **channels** to communicate between themselves and the main function:

1. A *stored-game* channel, that will pass along chunks of games in the []types.StoredGame format.
2. A *player-info* channel, that will pass along intermediate computations of player information in the simple types.PlayerInfo format.
3. A *done* channel, whose only purpose is to flag to the main function when all has been processed.

Each channel should also be able to pass an optional error, so tuples will be used.

The processing **routines** will be divided as per the following:

1. The **game loading** routine will:
   * Fetch all games in paginated arrays.
   * Send the separate arrays on the *stored-game* channel.
   * Send an error on the *stored-game* channel if any is encountered.
   * Close the *stored-game* channel after the last array, or on an error.
2. The **game processing** routine will:
   * Receive separate arrays of games from the *stored-game* channel.
   * Compute the aggregate player info records from them (i.e. ***map***).
   * Send the results on the *player-info* channel.
   * Pass along an error if it receives any.
   * Close the *player-info* channel after the last stored game, or on an error.
3. The **player info processing** routine will:
   * Receive individual player info records from the *player-info* channel.
   * Fetch the corresponding player info from the store. If it does not exist yet, it will create an empty new one.
   * Update the won and lost counts (i.e. ***reduce***). Remember, here it is doing += k, not += 1.
   * Save it back to the store.
   * Pass along an error if it receives any.
   * Close the *done* channel after the last player info, or on an error.
4. The **main function** will:
   * Create the above 3 channels.
   * Launch the above 3 routines.
   * Wait for the flag on the *done* channel.
   * Exit, perhaps with an error.

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

The processing will take your module's store from consensus version 2 to version 3. Therefore it makes sense to add the function in x/checkers/migrations/cv3/keeper.

The player info processing will handle an in-memory map of player addresses to their information: map[string]\*types.PlayerInfo. Create a new file to encapsulate this whole processing. Start by creating a helper that automatically populates it with empty values when information is missing:



Copy

func getOrNewPlayerInfoInMap(infoSoFar \*map[string]\*types.PlayerInfo, playerIndex string) (playerInfo \*types.PlayerInfo) {

playerInfo, found := (\*infoSoFar)[playerIndex]

if !found {

playerInfo = &types.PlayerInfo{

Index: playerIndex,

WonCount: 0,

LostCount: 0,

ForfeitedCount: 0,

}

(\*infoSoFar)[playerIndex] = playerInfo

}

return playerInfo

}

x /

checkers /

... /

keeper /

migration\_player\_info.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migrations/cv3/keeper/migration_player_info.go" \l "L13-L25" \t "_blank)

Now, create the function to load the games:



Copy

type storedGamesChunk struct {

StoredGames []types.StoredGame

Error error

}

func loadStoredGames(context context.Context,

k keeper.Keeper,

gamesChannel chan<- storedGamesChunk,

chunk uint64) {

defer func() { close(gamesChannel) }()

response, err := k.StoredGameAll(context, &types.QueryAllStoredGameRequest{

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

})

if err != nil {

gamesChannel <- storedGamesChunk{Error: err}

return

}

gamesChannel <- storedGamesChunk{StoredGames: response.StoredGame}

for response.Pagination.NextKey != nil {

response, err = k.StoredGameAll(context, &types.QueryAllStoredGameRequest{

Pagination: &query.PageRequest{

Key: response.Pagination.NextKey,

Limit: chunk,

},

})

if err != nil {

gamesChannel <- storedGamesChunk{Error: err}

return

}

gamesChannel <- storedGamesChunk{StoredGames: response.StoredGame}

}

}

x /

checkers /

... /

keeper /

migration\_player\_info.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migrations/cv3/keeper/migration_player_info.go" \l "L27-L58" \t "_blank)



Note that:

* The helper function passes along the channel a tuple storedGamesChunk 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.

Next, create the routine function to process the games:



Copy

type playerInfoTuple struct {

PlayerInfo types.PlayerInfo

Error error

}

func handleStoredGameChannel(k keeper.Keeper,

gamesChannel <-chan storedGamesChunk,

playerInfoChannel chan<- playerInfoTuple) {

defer func() { close(playerInfoChannel) }()

for games := range gamesChannel {

if games.Error != nil {

playerInfoChannel <- playerInfoTuple{Error: games.Error}

return

}

playerInfos := make(map[string]\*types.PlayerInfo, len(games.StoredGames))

for \_, game := range games.StoredGames {

var winner string

var loser string

if game.Winner == rules.PieceStrings[rules.BLACK\_PLAYER] {

winner = game.Black

loser = game.Red

} else if game.Winner == rules.PieceStrings[rules.RED\_PLAYER] {

winner = game.Red

loser = game.Black

} else {

continue

}

getOrNewPlayerInfoInMap(&playerInfos, winner).WonCount++

getOrNewPlayerInfoInMap(&playerInfos, loser).LostCount++

}

for \_, playerInfo := range playerInfos {

if playerInfo != nil {

playerInfoChannel <- playerInfoTuple{PlayerInfo: \*playerInfo}

}

}

}

}

x /

checkers /

... /

keeper /

migration\_player\_info.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migrations/cv3/keeper/migration_player_info.go" \l "L60-L96" \t "_blank)



Note that:

* This function can handle the edge case where black and red both refer to the same player.
* It prepares a map with a capacity equal to the number of games. At most the capacity would be double that. This is a value that could be worth investigating for best performance.
* Like the helper function, it 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.

Create the routine function to process the player info:



Copy

func handlePlayerInfoChannel(ctx sdk.Context, k keeper.Keeper,

playerInfoChannel <-chan playerInfoTuple,

done chan<- error) {

defer func() { close(done) }()

for receivedInfo := range playerInfoChannel {

if receivedInfo.Error != nil {

done <- receivedInfo.Error

return

}

existingInfo, found := k.GetPlayerInfo(ctx, receivedInfo.PlayerInfo.Index)

if found {

existingInfo.WonCount += receivedInfo.PlayerInfo.WonCount

existingInfo.LostCount += receivedInfo.PlayerInfo.LostCount

existingInfo.ForfeitedCount += receivedInfo.PlayerInfo.ForfeitedCount

} else {

existingInfo = receivedInfo.PlayerInfo

}

k.SetPlayerInfo(ctx, existingInfo)

}

done <- nil

}

x /

checkers /

... /

keeper /

migration\_player\_info.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migrations/cv3/keeper/migration_player_info.go" \l "L98-L123" \t "_blank)



Note that:

* This function only passes an optional error.
* It closes the channel it populates upon exit whether there is an error or not via the use of defer.

Now you can create the main function:



Copy

func MapStoredGamesReduceToPlayerInfo(ctx sdk.Context, k keeper.Keeper, chunk uint64) error {

context := sdk.WrapSDKContext(ctx)

gamesChannel := make(chan storedGamesChunk)

playerInfoChannel := make(chan playerInfoTuple)

done := make(chan error)

go handlePlayerInfoChannel(ctx, k, playerInfoChannel, done)

go handleStoredGameChannel(k, gamesChannel, playerInfoChannel)

go loadStoredGames(context, k, gamesChannel, chunk)

return <-done

}

x /

checkers /

... /

keeper /

migration\_player\_info.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migrations/cv3/keeper/migration_player_info.go" \l "L72-L106" \t "_blank)



Note that:

* The main function delegates the closing of channels to the routines.
* It starts the routines in the "reverse" order that they are chained, to reduce the likelihood of channel clogging.

Do not forget a suggested chunk size to pass as chunk uint64 to the main function when fetching stored games:



Copy

const (

ConsensusVersion = uint64(3)

+ StoredGameChunkSize = 1\_000

)

x /

checkers /

... /

types /

keys.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migrations/cv3/types/keys.go" \l "L5" \t "_blank)

To find the ideal chunk size value, you would have to test with the real state and try different values.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#unit-tests-2) Unit tests

You have added migration helpers and ought to add some unit tests on them. Similarly to other unit tests, you add a setup function in a new file:



Copy

func setupKeeperForV1ToV1\_1Migration(t testing.TB) (keeper.Keeper, context.Context) {

k, ctx := keepertest.CheckersKeeper(t)

checkers.InitGenesis(ctx, \*k, \*types.DefaultGenesis())

return \*k, sdk.WrapSDKContext(ctx)

}

Add a function that tests simple cases of storage:



Copy

func TestBuildPlayerInfosInPlace(t \*testing.T) {

tests := []struct {

name string

games []types.StoredGame

expected []types.PlayerInfo

}{

// TODO

}

for \_, tt := range tests {

for chunk := uint64(1); chunk < 5; chunk++ {

t.Run(fmt.Sprintf("%s chunk %d", tt.name, chunk), func(t \*testing.T) {

keeper, context := setupKeeperForV1ToV1\_1Migration(t)

ctx := sdk.UnwrapSDKContext(context)

for \_, game := range tt.games {

keeper.SetStoredGame(ctx, game)

}

cv3Keeper.MapStoredGamesReduceToPlayerInfo(ctx, keeper, chunk)

playerInfos := keeper.GetAllPlayerInfo(ctx)

require.Equal(t, len(tt.expected), len(playerInfos))

require.EqualValues(t, tt.expected, playerInfos)

})

}

}

}

Add the simple tests cases, such as:

1. Nothing:



Copy

{

name: "nothing to assemble",

games: []types.StoredGame{},

expected: []types.PlayerInfo(nil),

},

1. Single game with a win:



Copy

{

name: "single game with win",

games: []types.StoredGame{

{

Index: "1",

Winner: "b",

Black: "alice",

Red: "bob",

},

},

expected: []types.PlayerInfo{

{

Index: "alice",

WonCount: 1,

LostCount: 0,

},

{

Index: "bob",

WonCount: 0,

LostCount: 1,

},

},

},

And so on.

It can also be interesting to measure the time it takes to compute in the case of a large data set depending on the chunk size:



Copy

func TestBuild10kPlayerInfosInPlace(t \*testing.T) {

chunks := []uint64{1, 10, 100, 1\_000, 10\_000, 100\_000, 1\_000\_000}

for \_, chunk := range chunks {

keeper, context := setupKeeperForV1ToV1\_1Migration(t)

ctx := sdk.UnwrapSDKContext(context)

for id := uint64(1); id <= 100\_000; id++ {

keeper.SetStoredGame(ctx, types.StoredGame{

Index: strconv.FormatUint(id, 10),

Black: "alice",

Red: "bob",

Winner: "b",

})

}

before := time.Now()

cv3Keeper.MapStoredGamesReduceToPlayerInfo(ctx, keeper, chunk)

after := time.Now()

playerInfos := keeper.GetAllPlayerInfo(ctx)

require.Equal(t, 2, len(playerInfos))

require.EqualValues(t, []types.PlayerInfo{

{

Index: "alice",

WonCount: 100\_000,

},

{

Index: "bob",

LostCount: 100\_000,

},

}, playerInfos)

t.Logf("Chunk %d, duration %d millisec", chunk, after.Sub(before).Milliseconds())

}

}

You can run the tests with the verbose -v flag to get the log:

**Local**

**Docker**



Copy

$ go test -v github.com/alice/checkers/x/checkers/migrations/cv3/keeper

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

go test -v github.com/alice/checkers/x/checkers/migrations/cv3/keeper

Among the verbose test results, you can find something like:



Copy

=== RUN TestBuild10kPlayerInfosInPlace

migration\_player\_info\_test.go:153: Chunk 1, duration 2225 millisec

migration\_player\_info\_test.go:153: Chunk 10, duration 317 millisec

migration\_player\_info\_test.go:153: Chunk 100, duration 120 millisec

migration\_player\_info\_test.go:153: Chunk 1000, duration 103 millisec

migration\_player\_info\_test.go:153: Chunk 10000, duration 123 millisec

migration\_player\_info\_test.go:153: Chunk 100000, duration 172 millisec

migration\_player\_info\_test.go:153: Chunk 1000000, duration 158 millisec

--- PASS: TestBuild10kPlayerInfosInPlace (6.26s)

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-to-v1-1-migration-proper) v1 to v1.1 migration proper

The migration proper needs to execute the previous main function. You can encapsulate this knowledge in a function, which also makes more visible what is expected to take place:



Copy

package cv3

import (

"github.com/b9lab/checkers/x/checkers/keeper"

cv3Keeper "github.com/b9lab/checkers/x/checkers/migrations/cv3/keeper"

sdk "github.com/cosmos/cosmos-sdk/types"

)

func PerformMigration(ctx sdk.Context, k keeper.Keeper, storedGameChunk uint64) error {

ctx.Logger().Info("Start to compute checkers games to player info calculation...")

err := cv3Keeper.MapStoredGamesReduceToPlayerInfo(ctx, k, storedGameChunk)

if err != nil {

ctx.Logger().Error("Checkers games to player info computation ended with error")

} else {

ctx.Logger().Info("Checkers games to player info computation done")

}

return err

}

x /

checkers /

... /

cv3 /

migration.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/migrations/cv3/migration.go" \t "_blank)



This does not panic in case of an error. To avoid carrying on a faulty state, the caller of this function will have to handle the panic.

You have in place the functions that will handle the store migration. Now 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/2-migration-info.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/v0.45/core/upgrade.html) that are currently running. To signal an upgrade, your module needs to return a different value when queried by the upgrade module. You have already prepared this change from 2 to 3.



The consensus version number bears no resemblance to v1 or v1.1. The consensus version number is for the module, whereas v1 or v1.1 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). This is a name relevant at the application level. Keep this information in a sub-folder of app:



Copy

const (

UpgradeName = "v1tov1\_1"

)

app /

upgrades /

v1tov1\_1 /

keys.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/app/upgrades/v1tov1_1/keys.go" \l "L4" \t "_blank)

"v1tov1.1" would have been more elegant, but dots cause problems in governance proposal names.

You have to inform your app about:

1. The mapping between the consensus version(s) and the migration process(es).
2. The mapping between this name and the module(s) consensus version(s).

Prepare these in turn.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#callback-in-checkers-module) Callback in checkers module

Indicate that the checkers module needs to perform some upgrade steps when it is coming out of the old consensus version by calling RegisterMigration:



Copy

import (

...

+ cv2types "github.com/b9lab/checkers/x/checkers/migrations/cv2/types"

+ cv3 "github.com/b9lab/checkers/x/checkers/migrations/cv3"

cv3types "github.com/b9lab/checkers/x/checkers/migrations/cv3/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 cv3.PerformMigration(ctx, am.keeper, cv3types.StoredGameChunkSize)

+ }); err != nil {

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

+ }

}

x /

checkers /

module.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/x/checkers/module.go" \l "L146-L150" \t "_blank)

Note that:

* It decides on the chunk sizes to use at this point.
* It moves the consensus version up one version, from 2 to 3.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#callback-in-app) Callback in app

The function that you are going to write needs a Configurator. This is already created as part of your app preparation, but it is not kept. Instead of recreating one, adjust your code to make it easily available. Add this field to your app:



Copy

type App struct {

...

sm \*module.SimulationManager

+ configurator module.Configurator

}

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/app/app.go" \l "L238" \t "_blank)

Now adjust the place where the configurator is created:



Copy

- app.mm.RegisterServices(module.NewConfigurator(app.appCodec, app.MsgServiceRouter(), app.GRPCQueryRouter()))

+ app.configurator = module.NewConfigurator(app.appCodec, app.MsgServiceRouter(), app.GRPCQueryRouter())

+ app.mm.RegisterServices(app.configurator)

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/app/app.go" \l "L532-L533" \t "_blank)

Create a function that encapsulates knowledge about all possible upgrades, although there is a single one here. Since it includes *empty code* for future use, avoid cluttering the already long NewApp function:



Copy

import (

"github.com/b9lab/checkers/app/upgrades/v1tov1\_1"

storetypes "github.com/cosmos/cosmos-sdk/store/types"

)

func (app \*App) setupUpgradeHandlers() {

// v1 to v1.1 upgrade handler

app.UpgradeKeeper.SetUpgradeHandler(

v1tov1\_1.UpgradeName,

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

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

},

)

// When a planned update height is reached, the old binary will panic

// writing on disk the height and name of the update that triggered it

// This will read that value, and execute the preparations for the upgrade.

upgradeInfo, err := app.UpgradeKeeper.ReadUpgradeInfoFromDisk()

if err != nil {

panic(fmt.Errorf("failed to read upgrade info from disk: %w", err))

}

if app.UpgradeKeeper.IsSkipHeight(upgradeInfo.Height) {

return

}

var storeUpgrades \*storetypes.StoreUpgrades

switch upgradeInfo.Name {

case v1tov1\_1.UpgradeName:

}

if storeUpgrades != nil {

// configure store loader that checks if version == upgradeHeight and applies store upgrades

app.SetStoreLoader(upgradetypes.UpgradeStoreLoader(upgradeInfo.Height, storeUpgrades))

}

}

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/app/app.go" \l "L752-L783" \t "_blank)

Now you are ready to inform the app proper. You do this towards the end, after the call to app.SetEndBlocker and before if loadLatest. At the correct location:



Copy

...

app.SetEndBlocker(app.EndBlocker)

+ app.setupUpgradeHandlers()

if loadLatest {

...

}

app /

app.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/app/app.go" \l "L581" \t "_blank)



Be aware that the monitoring module added by Ignite causes difficulty when experimenting below with the CLI. To keep things simple, remove [all references to monitoring (opens new window)↗](https://github.com/cosmos/b9-checkers-academy-draft/compare/player-info-handling..player-info-migration#diff-0f1d2976054440336a576d47a44a37b80cdf6701dd9113012bce0e3c425819b7L159) from app.go.

When done right, adding the callbacks is a short and easy solution.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#integration-tests) Integration tests

With changes made in app.go, unit tests are inadequate – you have to test with integration tests. Take inspiration from the upgrade keeper's [own integration tests (opens new window)↗](https://github.com/cosmos/cosmos-sdk/blob/v0.45.4/x/upgrade/keeper/keeper_test.go#L215-L233).

In a new folder dedicated to your migration integration tests, copy the test suite and its setup function, which you created earlier for integration tests, minus the unnecessary checkersModuleAddress line:



Copy

type IntegrationTestSuite struct {

suite.Suite

app \*checkersapp.App

msgServer types.MsgServer

ctx sdk.Context

queryClient types.QueryClient

}

func TestUpgradeTestSuite(t \*testing.T) {

suite.Run(t, new(IntegrationTestSuite))

}

func (suite \*IntegrationTestSuite) SetupTest() {

app := checkersapp.Setup(false)

ctx := app.BaseApp.NewContext(false, tmproto.Header{Time: time.Now()})

app.AccountKeeper.SetParams(ctx, authtypes.DefaultParams())

app.BankKeeper.SetParams(ctx, banktypes.DefaultParams())

queryHelper := baseapp.NewQueryServerTestHelper(ctx, app.InterfaceRegistry())

types.RegisterQueryServer(queryHelper, app.CheckersKeeper)

queryClient := types.NewQueryClient(queryHelper)

suite.app = app

suite.msgServer = keeper.NewMsgServerImpl(app.CheckersKeeper)

suite.ctx = ctx

suite.queryClient = queryClient

}

tests /

integration /

... /

cv3 /

upgrade\_integration\_suite\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/tests/integration/checkers/migrations/cv3/upgrade_integration_suite_test.go" \t "_blank)

It is necessary to redeclare, as you cannot import test elements across package boundaries.

The code that runs for these tests is always at consensus version 3. After all, you cannot wish away the player info code during the tests setup. However, you can make the upgrade module believe that it is still at the old state. Add this step into the suite's setup:



Copy

app.BankKeeper.SetParams(ctx, banktypes.DefaultParams())

+ initialVM := module.VersionMap{types.ModuleName: cv2types.ConsensusVersion}

+ app.UpgradeKeeper.SetModuleVersionMap(ctx, initialVM)

tests /

integration /

... /

cv3 /

upgrade\_integration\_suite\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/tests/integration/checkers/migrations/cv3/upgrade_integration_suite_test.go" \l "L39-L40" \t "_blank)

Now you can add a test in another file. It verifies that the consensus version increases as saved in the upgrade keeper, when calling an upgrade with the right name.



Copy

func (suite \*IntegrationTestSuite) TestUpgradeConsensusVersion() {

vmBefore := suite.app.UpgradeKeeper.GetModuleVersionMap(suite.ctx)

suite.Require().Equal(cv2types.ConsensusVersion, vmBefore[types.ModuleName])

v1Tov1\_1Plan := upgradetypes.Plan{

Name: v1tov1\_1.UpgradeName,

Info: "some text here",

Height: 123450000,

}

suite.app.UpgradeKeeper.ApplyUpgrade(suite.ctx, v1Tov1\_1Plan)

vmAfter := suite.app.UpgradeKeeper.GetModuleVersionMap(suite.ctx)

suite.Require().Equal(cv3types.ConsensusVersion, vmAfter[types.ModuleName])

}

tests /

integration /

... /

cv3 /

upgrade\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/tests/integration/checkers/migrations/cv3/upgrade_test.go" \l "L19-L32" \t "_blank)

You can also confirm that it panics if you pass it a wrong upgrade name:



Copy

func (suite \*IntegrationTestSuite) TestNotUpgradeConsensusVersion() {

vmBefore := suite.app.UpgradeKeeper.GetModuleVersionMap(suite.ctx)

suite.Require().Equal(cv2types.ConsensusVersion, vmBefore[types.ModuleName])

dummyPlan := upgradetypes.Plan{

Name: v1tov1\_1.UpgradeName + "no",

Info: "some text here",

Height: 123450000,

}

defer func() {

r := recover()

suite.Require().NotNil(r, "The code did not panic")

suite.Require().Equal(r, "ApplyUpgrade should never be called without first checking HasHandler")

vmAfter := suite.app.UpgradeKeeper.GetModuleVersionMap(suite.ctx)

suite.Require().Equal(cv2types.ConsensusVersion, vmAfter[types.ModuleName])

}()

suite.app.UpgradeKeeper.ApplyUpgrade(suite.ctx, dummyPlan)

}

tests /

integration /

... /

cv3 /

upgrade\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/tests/integration/checkers/migrations/cv3/upgrade_test.go" \l "L34-L52" \t "_blank)

After that, you can check that the player infos are tallied as expected by adding in storage three completed games and one still in-play, and then triggering the upgrade:



Copy

func (suite \*IntegrationTestSuite) TestUpgradeTallyPlayerInfo() {

suite.app.CheckersKeeper.SetStoredGame(suite.ctx, types.StoredGame{

Index: "1",

Black: alice,

Red: bob,

Winner: rules.PieceStrings[rules.BLACK\_PLAYER],

})

suite.app.CheckersKeeper.SetStoredGame(suite.ctx, types.StoredGame{

Index: "2",

Black: alice,

Red: carol,

Winner: rules.PieceStrings[rules.RED\_PLAYER],

})

suite.app.CheckersKeeper.SetStoredGame(suite.ctx, types.StoredGame{

Index: "3",

Black: alice,

Red: carol,

Winner: rules.PieceStrings[rules.BLACK\_PLAYER],

})

suite.app.CheckersKeeper.SetStoredGame(suite.ctx, types.StoredGame{

Index: "4",

Black: alice,

Red: bob,

Winner: rules.PieceStrings[rules.NO\_PLAYER],

})

suite.Require().EqualValues([]types.PlayerInfo(nil), suite.app.CheckersKeeper.GetAllPlayerInfo(suite.ctx))

v1Tov1\_1Plan := upgradetypes.Plan{

Name: v1tov1\_1.UpgradeName,

Info: "some text here",

Height: 123450000,

}

suite.app.UpgradeKeeper.ApplyUpgrade(suite.ctx, v1Tov1\_1Plan)

expectedInfos := map[string]types.PlayerInfo{

alice: {

Index: alice,

LostCount: 1,

WonCount: 2,

},

bob: {

Index: bob,

LostCount: 1,

},

carol: {

Index: carol,

LostCount: 1,

WonCount: 1,

},

}

for who, expectedInfo := range expectedInfos {

storedInfo, found := suite.app.CheckersKeeper.GetPlayerInfo(suite.ctx, who)

suite.Require().True(found)

suite.Require().Equal(expectedInfo, storedInfo)

}

}

Expand



tests /

integration /

... /

cv3 /

upgrade\_test.go

[View source→](https://github.com/cosmos/b9-checkers-academy-draft/blob/player-info-migration/tests/integration/checkers/migrations/cv3/upgrade_test.go" \l "L54-L111" \t "_blank)

To run the tests, put the right package path:

**Local**

**Docker**



Copy

$ go test github.com/alice/checkers/tests/integration/checkers/migrations/cv3

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

go test github.com/alice/checkers/tests/integration/checkers/migrations/cv3

The tests confirm that you got it right.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#interact-via-the-cli) Interact via the CLI

You can already 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 code.
* Build the v1 checkers executable.
* Initialize a local blockchain and network.
* Run v1 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 checkers to halt on its own at the upgrade height.
* Check out the checkers v1.1 code.
* Build the v1.1 checkers executable.
* Run v1.1 checkers.
* Confirm that you now have a correct tally of player info.

Start your engines!

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#launch-v1) Launch v1

After committing your changes, in a shell checkout v1 of checkers with the content of the *run in production* work:



Copy

$ git checkout run-prod

$ git submodule update --init

Build the v1 executable for your platform:

**Local**

**Docker**



Copy

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

Copy

$ docker run --rm -it \

-v $(pwd):/checkers \

-w /checkers \

checkers\_i \

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

With the release/v1/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/checkersd keys add alice --keyring-backend test

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

Copy

$ docker network create checkers-net

$ docker create -it \

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

-p 26657:26657 \

--network checkers-net \

--name checkers \

checkers\_i

$ docker start checkers

$ docker exec -t checkers \

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

$ docker exec -t checkers \

./release/v1/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, reuse them all in the next calls.

Create a new genesis:

**Local**

**Docker**



Copy

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

Copy

$ docker exec -t checkers \

./release/v1/checkersd init checkers-1 --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/checkersd add-genesis-account \

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

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

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

Copy

$ docker exec -t checkers \

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

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

$ docker exec -t checkers \

./release/v1/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**

**By hand**



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"

Copy

...

- "voting\_period": "172800s"

+ "voting\_period": "600s"

...

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/checkersd gentx alice 100000000stake \

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

$ ./release/v1/checkersd collect-gentxs

Copy

$ docker exec -t checkers \

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

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

$ docker exec -t checkers \

./release/v1/checkersd collect-gentxs

Now you can start the chain proper:

**Local**

**Docker**



Copy

$ ./release/v1/checkersd start

Copy

$ docker exec -it checkers \

./release/v1/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/2-migration-info.html#add-games) Add games

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

**Local**

**Docker**



Copy

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

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

$ ./release/v1/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/checkersd keys show alice -a --keyring-backend test)

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

$ docker exec -t checkers \

./release/v1/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 expect a faucet to be available. Because that is not the case, you need to:

1. Skip the faucet calls by adjusting the "credit test accounts" before. Just return before [this.timeout (opens new window)↗](https://github.com/cosmos/academy-checkers-ui/blob/server-indexing/test/integration/stored-game-action.ts#L65).
2. 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/checkersd tx bank \

send $alice cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql 300stake \

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

--chain-id checkers-1 \

--broadcast-mode block --yes

$ ./release/v1/checkersd tx bank \

send $alice cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql 10token \

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

--chain-id checkers-1 \

--broadcast-mode block --yes

$ ./release/v1/checkersd tx bank \

send $bob cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22 300stake \

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

--chain-id checkers-1 \

--broadcast-mode block --yes

$ ./release/v1/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/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/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/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/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 a mix of complete and incomplete games:

**Local**

**Docker**



Copy

$ ./release/v1/checkersd query checkers \

list-stored-game --output json \

| jq '.storedGame[] | { "index":.index, "winner":.winner }'

Copy

$ docker exec -t checkers \

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

list-stored-game --output json \

| jq '.storedGame[] | { \"index\":.index, \"winner\":.winner }'"

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/2-migration-info.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/checkersd status \

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

Copy

$ docker exec -t checkers \

bash -c './release/v1/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

Submit your governance proposal upgrade:

**Local**

**Docker**



Copy

$ ./release/v1/checkersd tx gov submit-proposal software-upgrade v1tov1\_1 \

--title "v1tov1\_1" \

--description "Introduce a tally of games per player" \

--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/checkersd tx gov submit-proposal software-upgrade v1tov1\_1 \

--title "v1tov1\_1" \

--description "Introduce a tally of games per player" \

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

--chain-id checkers-1 \

--broadcast-mode block \

--upgrade-height 1180 \

--deposit 10000000stake

This returns something with:



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/checkersd tx gov vote 1 yes \

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

--chain-id checkers-1

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

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

--chain-id checkers-1

Copy

$ docker exec -t checkers \

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

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

--chain-id checkers-1

$ docker exec -t checkers \

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

Copy

$ docker exec -t checkers \

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

Copy

$ docker exec -t checkers \

./release/v1/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 "v1tov1\_1" NEEDED at height: 1180:

6:29PM ERR CONSENSUS FAILURE!!! err="UPGRADE \"v1tov1\_1\" 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/checkersd status \

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

Copy

$ docker exec -it checkers \

bash -c './release/v1/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":"v1tov1\_1","height":1180}

With your node (and therefore your whole blockchain) down, you are ready to move to v1.1.

[#Copy link](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#launch-v1-1) Launch v1.1

With v1 stopped and its state saved, it is time to move to v1.1. Checkout v1.1 of checkers, for instance:



Copy

$ git checkout player-info-migration

Back in the first shell, build the v1.1 executable:

**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

Launch it:

**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"

It should start and display something like:



Copy

...

7:06PM INF applying upgrade "v1tov1\_1" at height: 1180

7:06PM INF migrating module checkers from version 2 to version 3

7:06PM INF Start to compute checkers games to player info calculation...

7:06PM INF Checkers games to player info computation done

...

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

**Local**

**Docker**



Copy

$ ./release/v1\_1/checkersd query checkers list-player-info

Copy

$ docker exec -t checkers \

./release/v1\_1/checkersd query checkers list-player-info

This should print something like:



Copy

playerInfo:

- forfeitedCount: "0"

index: cosmos1fx6qlxwteeqxgxwsw83wkf4s9fcnnwk8z86sql

lostCount: "0"

wonCount: "3"

- forfeitedCount: "0"

index: cosmos1mql9aaux3453tdghk6rzkmk43stxvnvha4nv22

lostCount: "3"

wonCount: "0"

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 almost done! It now needs a leaderboard, which is introduced in the next section.

synopsis

To summarize, this section has explored:

* How to add a new data structure in storage as a breaking change.
* How to upgrade a blockchain in production, by migrating from v1 of the blockchain to v1.1, and the new data structures 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 private helper functions.
* Worthwhile unit tests with regard to player info handling.
* Integration tests to further confirm the validity of the upgrade.
* A complete procedure for how to conduct the update via the CLI.

previous

[](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)**

up next

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

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

[Introducing a new data structure](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#introducing-a-new-data-structure)

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

[New v1.1 information](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#new-v1-1-information)

[v1.1 player information helpers](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-1-player-information-helpers)

[v1.1 player information handling](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-1-player-information-handling)

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

[Player info handling unit tests](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#player-info-handling-unit-tests)

[v1 to v1.1 player information migration helper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-to-v1-1-player-information-migration-helper)

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

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

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

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

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

[v1 to v1.1 migration proper](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#v1-to-v1-1-migration-proper)

[Consensus version and name](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#consensus-version-and-name)

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

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

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

[Interact via the CLI](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#interact-via-the-cli)

[Launch v1](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#launch-v1)

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

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

[Launch v1.1](https://ida.interchain.io/hands-on-exercise/4-run-in-prod/2-migration-info.html#launch-v1-1)

#### **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)