



Exploring The Demo

[Canion Network Quickstart Guide | 2025](#)

Version: 1.0.8-2025-05-16

Contents

[Exploring the Demo](#)

[Prerequisites](#)

[Walkthrough](#)

[Canton Console](#)

[Daml Shell](#)

[Connect to DevNet](#)

[Important: Migration ID for DevNet Connections](#)

[Configuring Non-Default DevNet Sponsors](#)

[SV UIs](#)

[Canton Coin Scan](#)

[Observability Dashboard](#)

[Development Journey in the CN QS Lifecycle](#)

[CN QS Components](#)

[Development Tools](#)

[LocalNet](#)

[Network Components](#)

[ScratchNet](#)

[Sample Application](#)

[Application Components](#)

[Development Lifecycle](#)

[Learning Phase](#)

[Experimentation Phase](#)

[Development Phase](#)

[Gradle Settings](#)

[Environment Variables](#)

[Docker Compose](#)

[Separation Phase](#)

[Ongoing Updates](#)

[Upgrades On The Global Synchronizer](#)

[Type 1: Backward-Compatible Changes](#)

[Type 2: Daml Model Changes](#)

[Type 3: Non-Compatible Protocol Changes](#)

[Preparing for Upgrades](#)

[Keycloak in the CN QS](#)

[Realm Structure](#)

[Keycloak Configuration](#)

[Customizing Keycloak for Business Needs](#)

[Accessing the Admin Console](#)

[Customization Scenarios](#)

[Add a New User](#)

[Modify Client Settings](#)

[Add a New Client](#)

[Update Environment Variables](#)

[Troubleshooting](#)

[Next Steps](#)

Exploring the Demo

The CN QS and its guides are a work-in-progress (WIP). As a result, the CN QS guides may not be a little out of step with the application. If you find errors or other inconsistencies, please contact your representative at Digital Asset.

This section works through a complete business operation within the CN QS.

Prerequisites

You should have successfully [installed the CN QS](#) before beginning this demonstration.

Access to the [CN-Quickstart Github repository](#) and [CN Docker repository](#) is needed to successfully pull the Digital Asset artifacts from JFrog Artifactory.

Access to the *Daml-VPN* connection or [a SV Node](#) that is whitelisted on the CN is required to connect to DevNet. The GSF publishes a [list of SV nodes](#) who have the ability to sponsor a Validator node. To access DevNet, contact your sponsoring SV agent for VPN connection information.

If you need support accessing the SV or VPN email support@digitalasset.com.

The CN QS is a Dockerized application and requires [Docker Desktop](#). It is recommended to allocate 8 GB of memory and 3 GB of Swap memory to properly run the required Docker containers. If you witness unhealthy containers, please consider allocating additional resources, if possible.

DevNet is less intensive because the SVs and other LocalNet containers are hosted outside of your local machine.

Walkthrough

After the QS is installed and running, confirm that you are in the quickstart subdirectory of the CN QS.

Open an incognito browser.

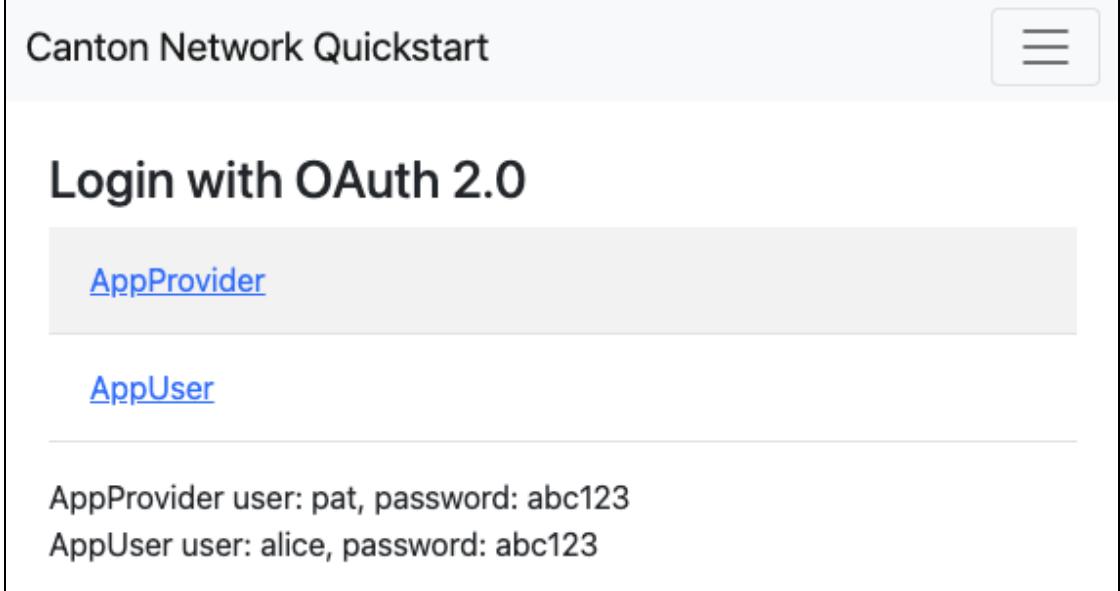
Navigate to:

localhost:3000/login

 Currently, localhost URLs do not work in Safari. We are working on a solution and apologize for the inconvenience.

Alternatively, in the terminal, from quickstart/ run:

```
make open-app-ui
```

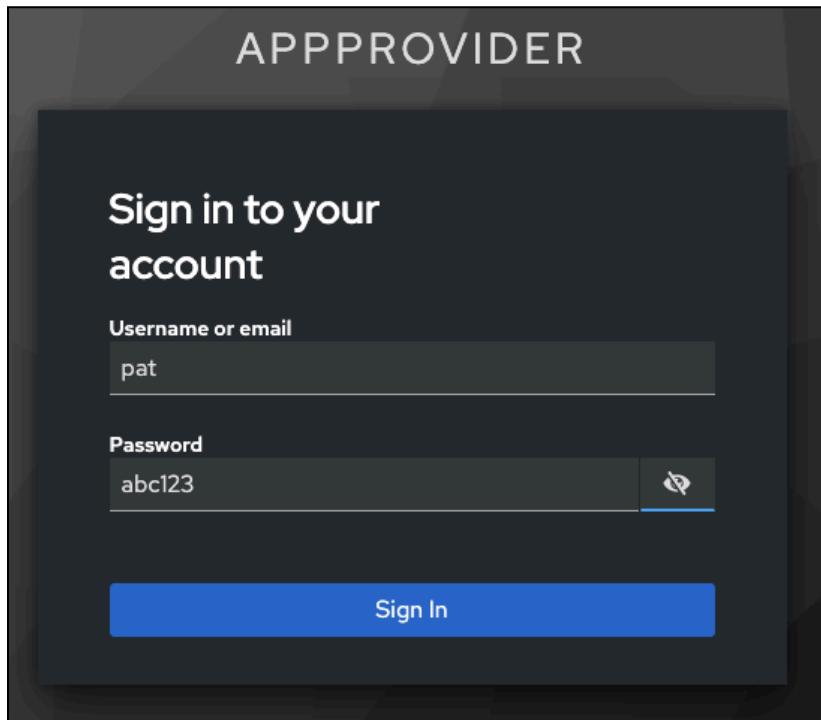


The screenshot shows a web-based application titled "Canton Network Quickstart". At the top right is a menu icon. Below the title, the text "Login with OAuth 2.0" is displayed. Two buttons are present: "AppProvider" and "AppUser". Underneath these buttons, two sets of credentials are listed: "AppProvider user: pat, password: abc123" and "AppUser user: alice, password: abc123".

Make note that the AppProvider's username is “pat” and the password is “abc123” (all lowercase).

Login as the AppProvider.

Fill in the login credentials: username: pat, password: abc123



Select “AppInstalls” in the menu.

A screenshot of the "Canton Network Quickstart" application. The top navigation bar includes links for "Canton Network Quickstart", "Home", "AppInstalls", "Licenses", and "Tenants". On the right, there is a link "Pat the provider". The main content area is titled "App Installs". A note at the top of this section says "Note: Run make create-app-install-request to submit an AppInstallRequest". Below this, a table is displayed with columns: Contract ID, Status, DSO, Provider, User, Meta, # Licenses, and Actions.

Open a terminal.

From /quickstart/ run:

```
make create-app-install-request
```

This command creates an App Installation Request on behalf of the Participant.

```
[(base) quickstart ~ % make create-app-install-request
docker compose -f docker/app-user-shell/compose.yaml --env-file .env run --rm create-app-install-request || true
get_token ledger-api-user AppProvider
get_user_party AppProvider participant-app-provider
http://participant-app-provider:7575/v2/users/AppProvider
get_token ledger-api-user Org1
get_user_party Org1 participant-app-user
http://participant-app-user:7575/v2/users/Org1
get_token administrator Org1
http://validator-app-user:5003/api/validator/v0/scan-proxy/dso-party-id
http://participant-app-user:7575/v2/commands/submit-and-wait
--data-raw {
  "commands" : [
    { "CreateCommand" : {
        "template_id": "#quickstart-licensing:Licensing.AppInstall:AppInstallRequest",
        "create_arguments": {
          "dso": "DSO::12209a3af80af3fa93853be8a8b9f5887055edc3b0a94f4b198f486d08d197784c09",
          "provider": "AppProvider::1220b3de80de523473aa2ca56745ef1fb29a2203dfd13029e0448336bbcf3
82aaaf86",
          "user": "Org1::1220e69bc6115cd8ed360aa6caafce122c2a24561262c2f6ce6a6070e170e9e8244d",
          "meta": {"values": []}
        }
      }
    ]
  },
  "workflow_id" : "create-app-install-request",
  "application_id": "ledger-api-user",
  "command_id": "create-app-install-request",
  "deduplication_period": { "Empty": {} },
  "act_as": ["Org1::1220e69bc6115cd8ed360aa6caafce122c2a24561262c2f6ce6a6070e170e9e8244d"],
  "read_as": ["Org1::1220e69bc6115cd8ed360aa6caafce122c2a24561262c2f6ce6a6070e170e9e8244d"],
  "submission_id": "create-app-install-request",
  "disclosed_contracts": [],
  "domain_id": "",
  "package_id_selection_preference": []
}
{"update_id":"1220aebcd64fc960c2aec834d41596789994f21a69ef9cab5639b8521169a0ca67","completion_offset":81}
```

If your machine is not powerful enough to host LocalNet or if the docker containers are not responsive then the response may show a failure with status code 404 or 000.

Increasing Docker memory limit to at least 8 GB should allow the LocalNet containers to operate properly.

```
[(base) quickstart ~ % make create-app-install-request
docker compose -f docker/app-user-shell/compose.yaml --env-file .env run --rm create-app-install-request || true
[+] Building 0.s (0/0)
[+] Building 0.s (0/0)
get_token ledger-api-user AppProvider
get_user_party AppProvider participant-app-provider
http://participant-app-provider:7575/v2/users/AppProvider
get_token ledger-api-user Org1
get_user_party Org1 participant-app-user
http://participant-app-user:7575/v2/users/Org1
get_token administrator Org1
http://validator-app-user:5003/api/validator/v0/scan-proxy/dso-party-id
Request failed with HTTP status code 404
Response body: The requested resource could not be found.
```

Return to the browser.

The install request appears in the list.

Click “Accept”.

Canton Network Quickstart Home AppInstalls Licenses Tenants Pat the provider

App Installs

Note: Run `make create-app-install-request` to submit an AppInstallRequest

Contract ID	Status	DSO	Provider	User	Meta	# Licenses	Actions
00a2f3c34cc5e...	REQUEST	DSO::12206b3c...	app_provider_q...	app_user_quick...	{"data":{}}	0	<button>Accept</button> <button>Reject</button> <button>Cancel</button>

The AppInstallRequest is Accepted. The actions update to create or cancel the license.

Canton Network Quickstart Home AppInstalls Licenses Tenants Pat the provider

App Installs

Note: Run `make create-app-install-request` to submit an AppInstallRequest

Success

```
Success: Accepted AppInstallRequest
00a2f3c34cc5edf660aeb6658cd7000541349f3e
5b2e3268330f1f3307067aa9fbca012204022da1
3d54f9c9321e2401364a3123671ebcadde47deb6
2ee39f9c81e4f0c9f
```

Contract ID	Status	DSO	Provider	User	Meta	# Licenses	Actions
00502121cb22e...	INSTALL	DSO::12206b3c...	app_provider_q...	app_user_quick...	{"data":{}}	0	<button>Create License</button> <button>Cancel Install</button>

Click “Create License”.

The license is created and the “# Licenses” field is updated.

Canton Network Quickstart Home AppInstalls Licenses Tenants Pat the provider

App Installs

Note: Run `make create-app-install-request` to submit an AppInstallRequest

Success

```
Success: Created License:
00be4bac78d731505d04d5e4964c9a297d9e479
54942c54809bc24d3b65cd0342bca010220a263
af0d31f1043cbb903c58b0479db70935d107ae0a
0a6299e510c84c402909
```

Contract ID	Status	DSO	Provider	User	Meta	# Licenses	Actions
00fdf01728922...	INSTALL	DSO::12206b3c...	app_provider_q...	app_user_quick...	{"data":{}}	1	<button>Create License</button> <button>Cancel Install</button>

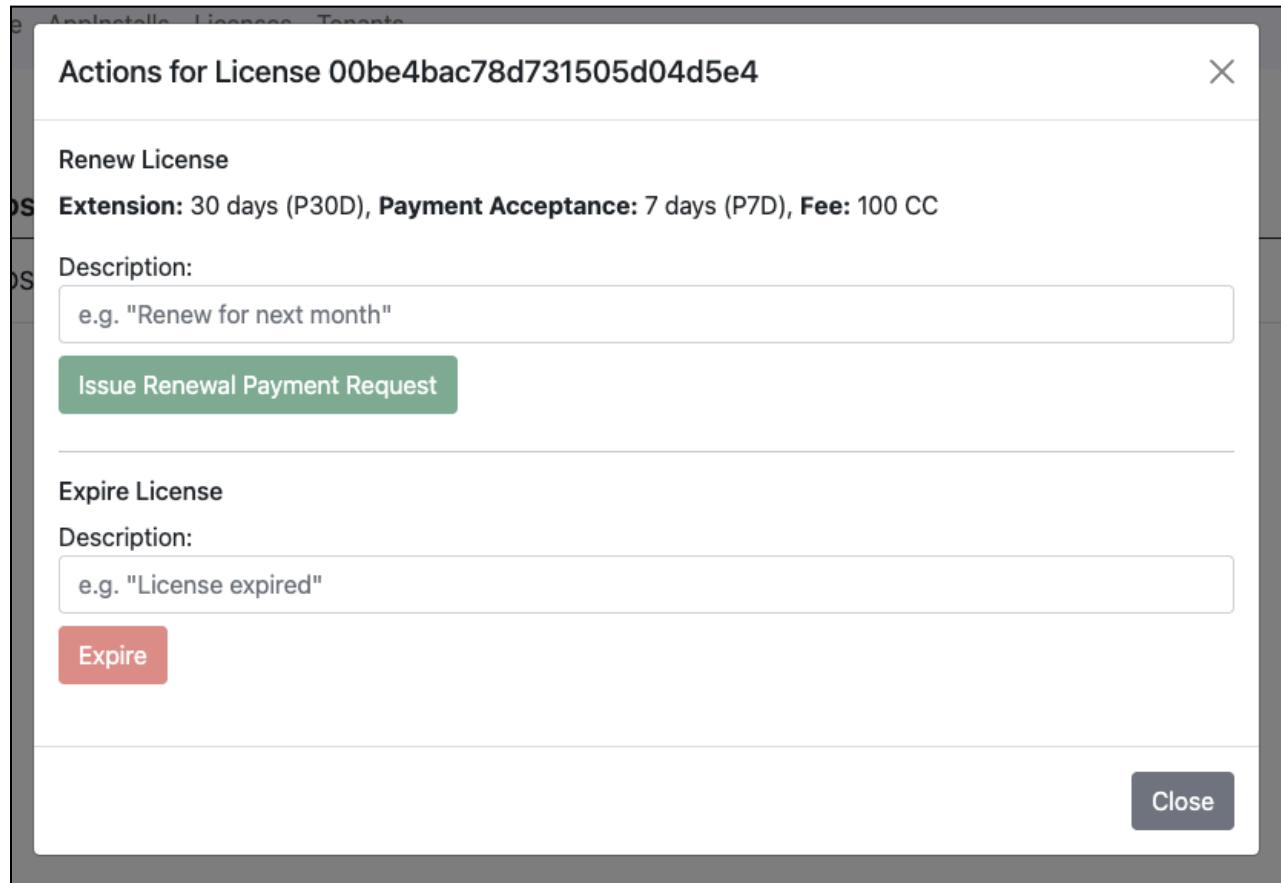
In the AppProvider, “Pat the provider’s,” account, navigate to the **Licenses** menu and select “Actions.”

Canton Network Quickstart Home AppInstalls Licenses Tenants Pat the provider

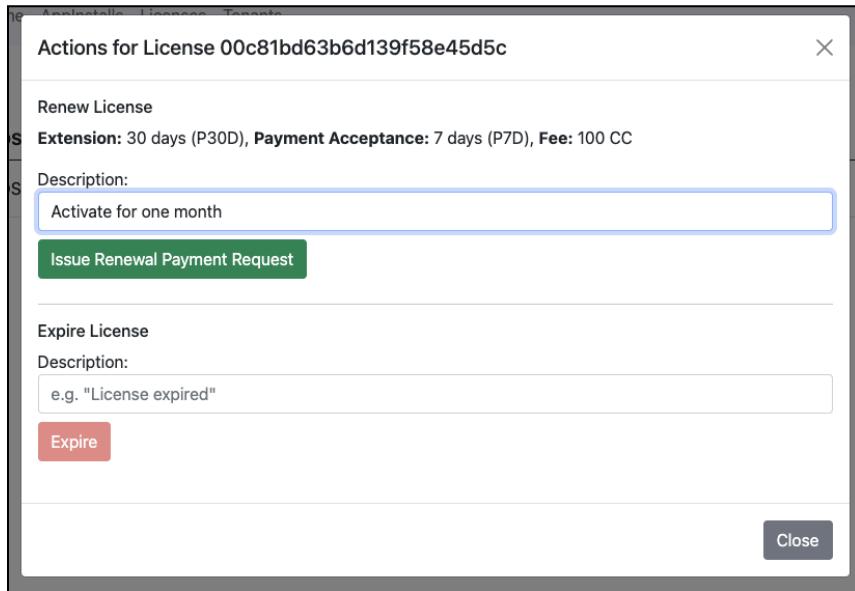
Licenses

License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions
00be4bac78d731505d04...	DSO::122...	app_provide...	app_user_q...	2025-03-13T21:20:...	1			<button>Actions</button>

An “Actions for License” modal opens with an option to renew or expire the license. Per the Daml contract, licenses are created in an expired state. To activate the license, it must be renewed.



To renew the license, enter a description then click the green “Issue Renewal Payment Request” button.



The license renewal process is initiated and ultimately successful.

Licenses								
License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions
00be4bac78d731505d04...	DSO::122...	app_provide...	app_user_q...	2025-03-13T21:20:...	1	100	30 days	<button>Actions</button>

A green success message box is displayed above the table, stating: "Success: License Renewal initiated successfully".

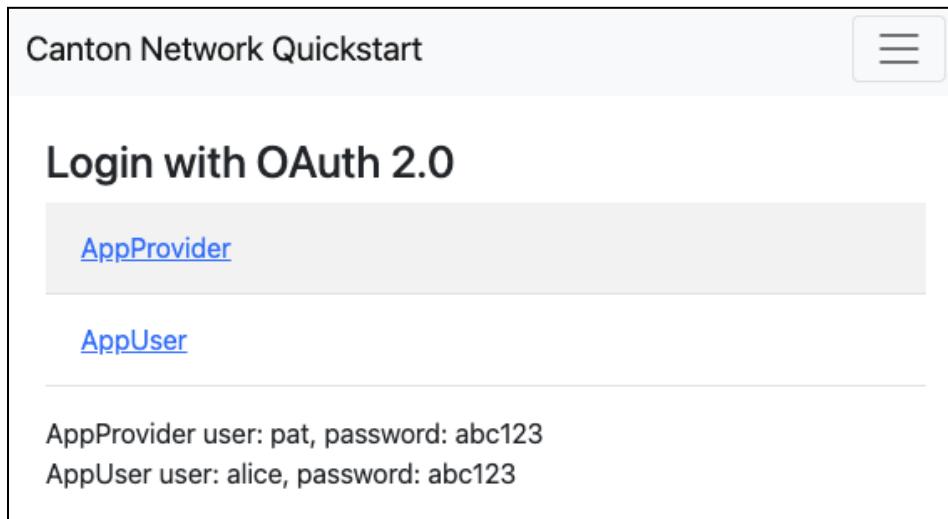
The license is now available for a 30-day extension for a flat fee of \$100 CC.

Licenses								
License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions
00be4bac78d731505d04...	DSO::122...	app_provide...	app_user_q...	2025-03-13T21:20:...	1	100	30 days	<button>Actions</button>

Pat the provider has done as much as they are able until Alice pays the renewal fee.

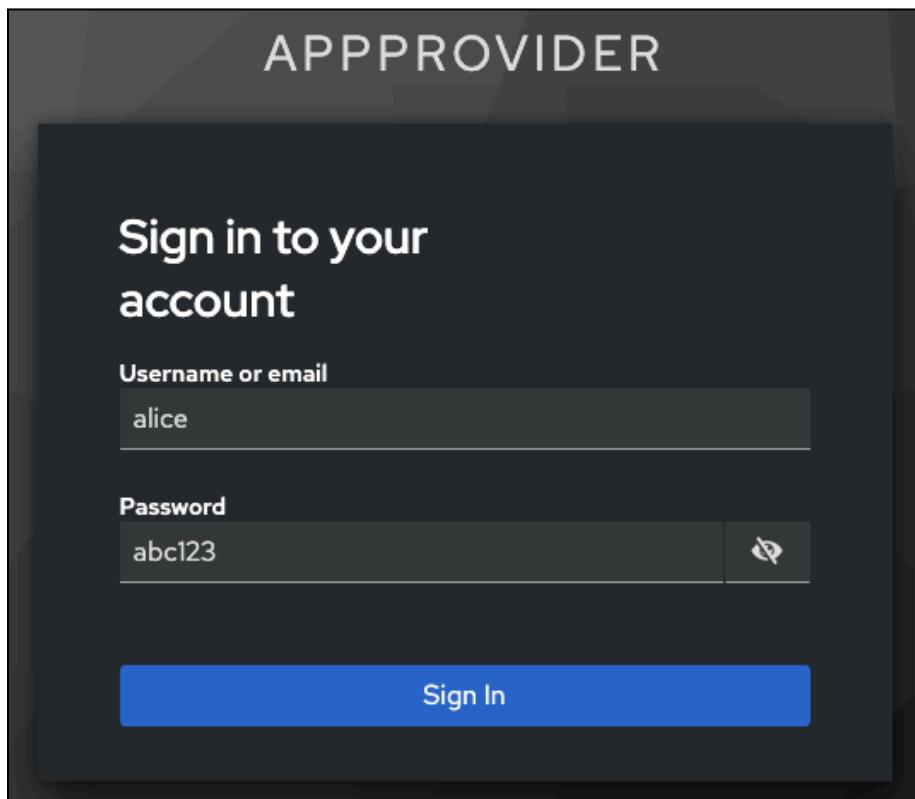
For the next step we recommend opening a separate browser in incognito mode. Each user, AppProvider, and Org1, should be logged into separate browsers for most consistent results. For example, if you logged into AppProvider using Chrome, you would use Firefox when logging into Org1.

Navigate to `http://localhost:3000/login` using a separate browser in incognito or private mode.



Login as AppUser alice.

Note that AppUser's username is "alice" and the password is "abc123".

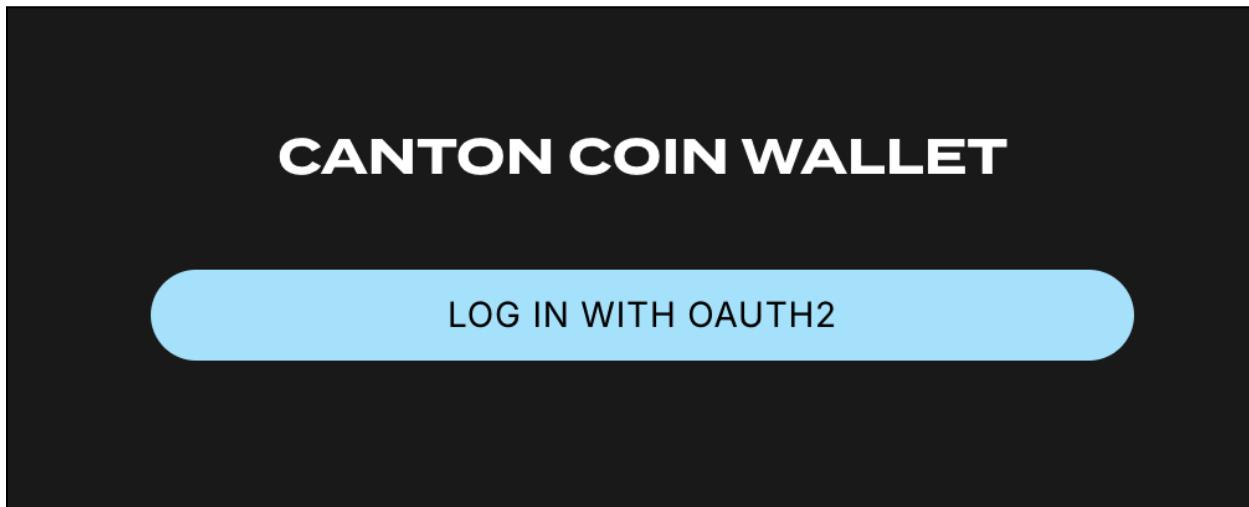


Go to the **Licenses View** and click the "Pay renewal" button.

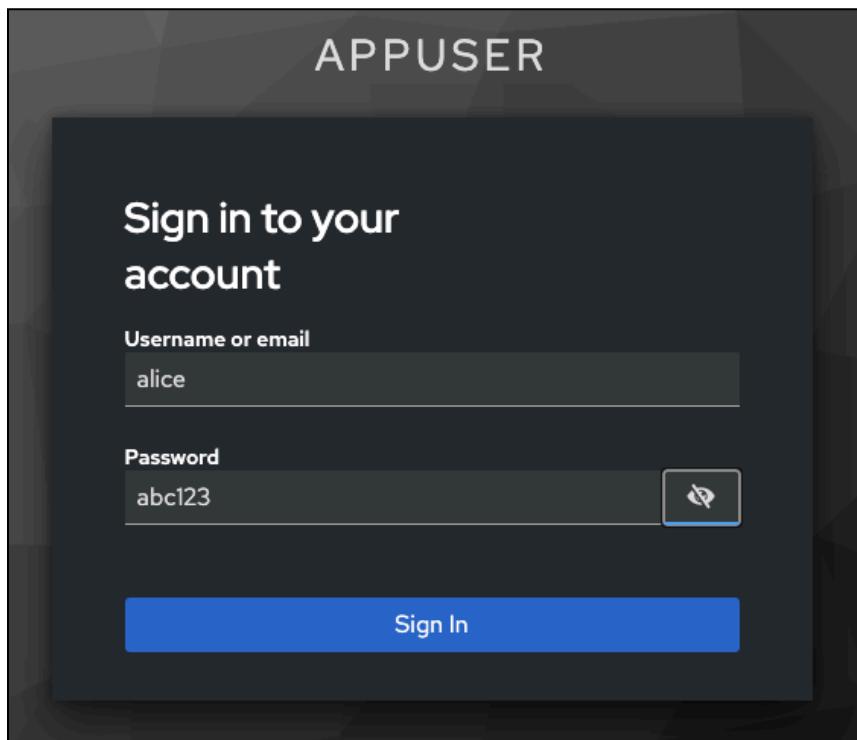
Canton Network Quickstart									Alice the user
Licenses									
License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions	
00be4bac78d731505d04...	DSO::122...	app_provide...	app_user_q...	2025-03-13T21:20:...	1	100	30 days	<button>Pay Renewal</button>	

Click on the Pay Renewal button. This navigates to the Canton Coin Wallet log in. Click “LOG IN WITH OAUTH2”.

 If you have any issues with log in, navigate directly to <http://wallet.localhost:2000/>.



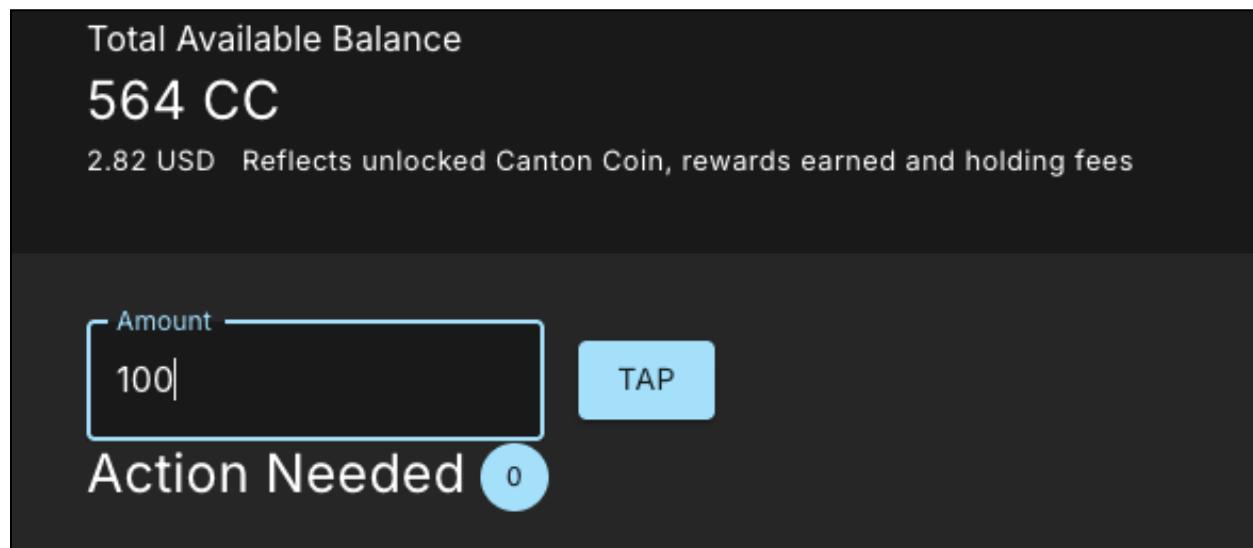
This navigates to a keycloak login.
Enter the same username and password as before.



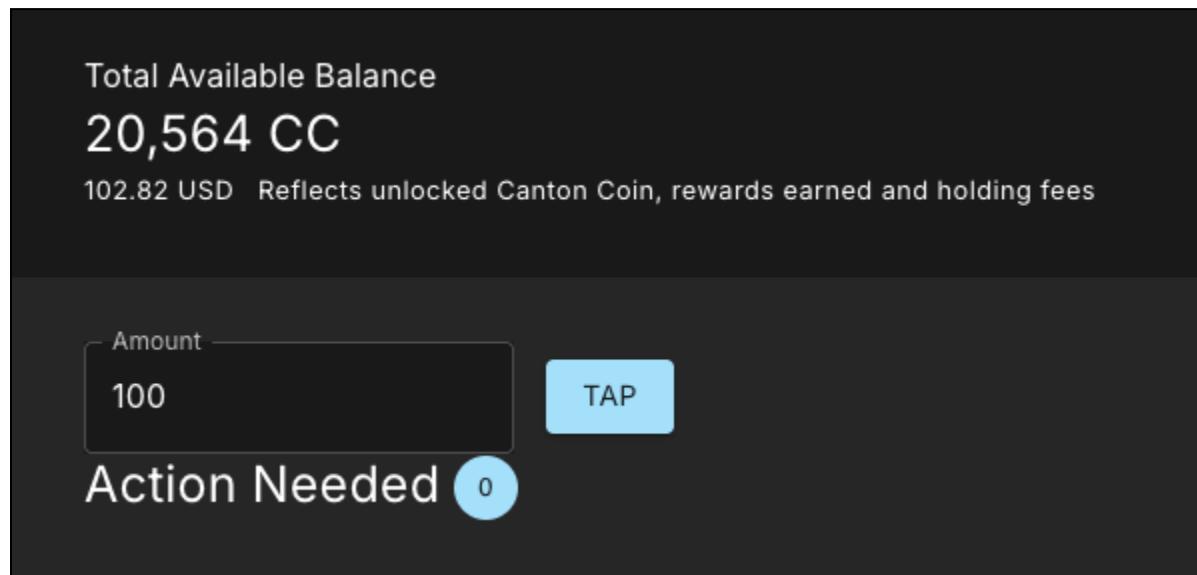
Signing in directs to the Canton Coin Wallet.

The wallet must be populated with CC in order to fulfill the transaction.

In CC Wallet, populate the wallet with \$100 USD, or the equivalent of 20,000 CC.



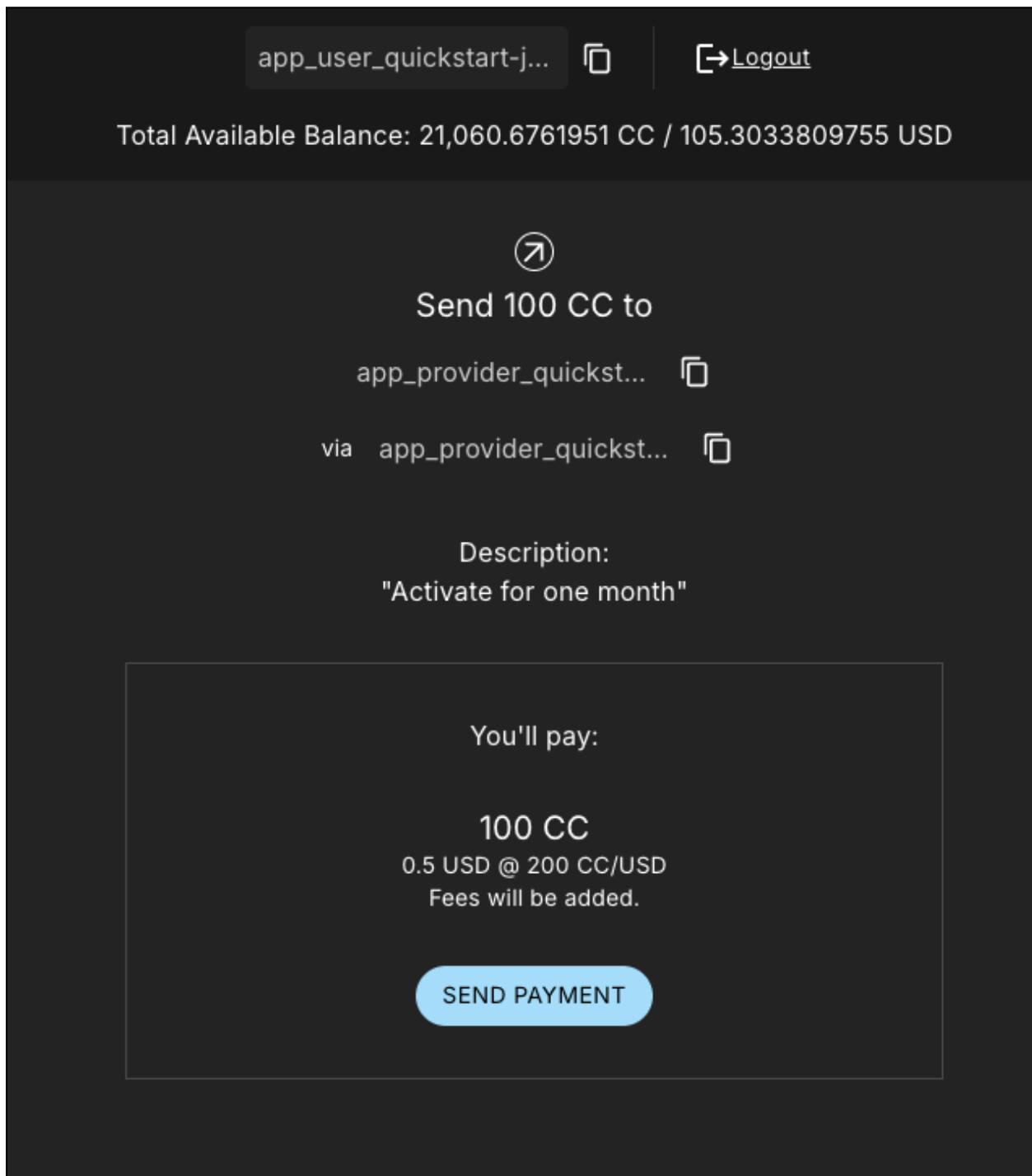
The wallet was prepopulated with 564 CC so it now contains 20,564 CC.



Return to the License Renewal Request as Org1. Click “Pay Renewal”.

Licenses								
License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions
00a01c1718dbe1a0f5e45...	DSO::122...	app_provide...	app_user_q...	2025-03-14T14:36:...	1	100	30 days	<button>Pay Renewal</button>

The CC Wallet balance is sufficient to send payment to the Provider.



Return to the AppProvider's License Renewal Requests View.
The AppProvider may now Complete the Renewal.

License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions
00a01c1718dbe1a0f5e45...	DSO::122...	app_provide...	app_user_q...	2025-03-14T14:36:...	1	100	30 days	<button>Complete Renewal</button>

Clicking "Complete Renewal" results in a Success.

License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions
00a01c1718dbe1a0f5e45...	DSO::122...	app_provide...	app_user_q...	2025-03-14T14:36:...	1			<button>Actions</button>

Alice's License view shows the activated license.

License Contract ID	DSO	Provider	User	Expires At	License #	Renew Fee	Extension	Actions
00f77a38c88b2ce54687...	DSO::122...	app_provide...	app_user_q...	2025-04-13T15:11:1...	1			<button>Actions</button>

Congratulations. You've successfully created and activated a license with a payment transfer!

Colton Console

The Colton Console connects to the running application ledger. The console allows a developer to bypass the UI to interact with the CN in a more direct manner. For example, in Colton Console you can connect to the Participant to see the location of the Participant and their synchronizer domain.

The app provider and the app user each have their own console. To activate the app provider's Colton Console in a terminal from the `quickstart/` directory. Run:

```
make console-app-provider
```

Open the participant's Colton Console with

```
make console-app-user
```

After the console initiates, run the `participant` and `participant.domains` commands, respectively.

`participant`

Returns their location in the ledger.

```
[@ participant
res0: com.digitalasset.canton.console.RemoteParticipantReference = Participant 'participant'
```

`participant.domains`

Shows the Participant's synchronizer.

```
[@ participant.domains
res1: participant.domains.type = com.digitalasset.canton.console.commands.ParticipantAdministration$domains$04f4c3934
```

`participant.health.ping(participant)`

Runs a health ping. The ping makes a round trip through the CN blockchain. Pinging yourself validates communication throughout the entire network.

```
[@ participant.health.ping(participant)
res0: Duration = 4979 milliseconds
```

Daml Shell

The Daml Shell connects to the running PQS database of the application provider's Participant. In the Shell, the assets and their details are available in real time.

Run the shell from `quickstart/` in the terminal with:

```
make shell
```

Run the following commands to see the data:

```
active
```

Shows unique identifiers and the asset count

```
postgres-splice-app-provider:5432/scribe> active
```

Identifier	Type	Count
quickstart-licensing:Licensing.AppInstall:AppInstall	Template	1
quickstart-licensing:Licensing.License:License	Template	1
splice-amulet:Splice.Amulet:Amulet	Template	1
splice-amulet:Splice.Amulet:ValidatorRight	Template	1
splice-wallet:Splice.Wallet.Install:WalletAppInstall	Template	1

```
active quickstart-licensing:Licensing.License:License
```

List the license details.

```
postgres-splice-app-provider:5432/scribe 6d → f7> active quickstart-licensing:Licensing.License:License
```

Created at	Contract ID	Contract Key	Payload
a9	000cd68ca7d2cf95454b...		dso: DS0::12203a329668884fac6377f41c924df6a11c59f3be909737d27799252930e537c42b user: Org1::12209d2965deec586b4a6d12b80e535bb52407fad54dc2dd88575291780ed5fd9ff4 params: meta: values: provider: AppProvider::12206e5249b12cd9fd05e9b25894c0663b73a18c90baccd7f2d48e7958157b510358 expiresAt: 2025-03-16T21:59:38.403435Z licenseNum: 1

```
active quickstart-licensing:Licensing.License:LicenseRenewalRequest
```

Displays license renewal request details.

```
archives quickstart-licensing:Licensing.AppInstall:AppInstallRequest
```

Shows any archived license(s).

```
postgres-splice-app-provider:5432/scribe 6d → f7> archives quickstart-licensing:Licensing.AppInstall:AppInstallRequest
```

Created at	Archived at	Contract ID	Contract Key	Payload
6f	75	00e906b2720a9b7965a3...		dso: DS0::12203a329668884fac6377f41c924df6a11c59f3be909737d27799252930e537c42b meta: values: user: Org1::12209d2965deec586b4a6d12b80e535bb52407fad54dc2dd88575291780ed5fd9ff4 provider: AppProvider::12206e5249b12cd9fd05e9b25894c0663b73a18c90baccd7f2d48e7958157b510358

Connect to DevNet

Stop the LocalNet containers to change the connection from LocalNet to DevNet.

In the terminal, run:

```
make stop && make clean-all
```

To edit the connection and observability parameters run:

```
make setup
```

When prompted to enable LocalNet, enter “n”. This enables DevNet

Optionally, enter “Y” to enable observability. This starts additional containers which may require more memory for Docker.

You may leave the party hint as the default value by tapping ‘return’ on the keyboard.

```
(base) quickstart ~ % make setup
Starting local environment setup tool...
./gradlew configureProfiles --no-daemon --console=plain --quiet
Enable LocalNet? (Y/n): n
  LOCALNET_ENABLED set to 'false'.

Enable Observability? (Y/n): Y
  OBSERVABILITY_ENABLED set to 'true'.

Specify a party hint (this will identify the participant in the network) [quickstart-... -1]:
  PARTY_HINT set to 'quickstart-... -1'.

.env.local updated successfully.
(base) quickstart ~ %
```

 Running `make setup` regenerates `.env.local` but preserves the contents of the `.env` file settings.

The application is now connected to DevNet.

Important: Migration ID for DevNet Connections

When connecting to DevNet, verify that the `MIGRATION_ID` value in `.env` matches the current network migration ID for your DevNet Super Validator (SV).

Check the current migration ID at <https://sync.global/sv-network/> under the GSF DevNet information section.

For example, if the SV Node Information shows the `migration_id` value as “0” then update `MIGRATION_ID` to “0” in your `.env`.

GSF DevNet Super Validator Node Information

```
{  
  "network": "devnet",  
  "sv": {  
    "migration_id": 0,  
    "version": "0.3.15"  
  },  
  "synchronizer": {  
    "active": {  
      "chain_id_suffix": "5",  
      "migration_id": 0,  
      "version": "0.3.15"  
    },  
    "legacy": null,  
    "staging": null  
  }  
}
```

In .env:

```
ONBOARDING_SECRET_URL=https://sv.sv-1.dev.global.canton.network.digitalasset.com/api/sv/v0/devnet/onboard/validator/prepare  
MIGRATION_ID=0  
APP_PROVIDER_VALIDATOR_PARTICIPANT_ADDRESS=participant-app-provider  
APP_USER_VALIDATOR_PARTICIPANT_ADDRESS=participant-app-user
```

Configuring Non-Default DevNet Sponsors

In DevNet mode, you can configure a non-default SPONSOR_SV_ADDRESS, SCAN_ADDRESS and ONBOARDING_SECRET_URL or ONBOARDING_SECRET in the quickstart/.env file.

 Connecting to DevNet requires a connection to an [approved SV](#). If your organization provides access to the DAML-VPN, then connect to it to access the Digital Asset-sponsored SV.

Your organization may sponsor another [CN-approved SV](#). If this is the case, speak with your administrator for privileged access.

Review the DevNet Global Synchronizer documentation to learn more about the [SV onboarding process](#).

 If you run into errors when making DevNet operations, double check that the DevNet VPN is active. DevNet VPNs may timeout, especially if left unattended for extended periods of time.

In an incognito browser navigate to `localhost:3000/login`. Login as the `Org1 user` and create and archive assets, as before. Logout and do the same as the `AppProvider`.

Canton Coin Scan

While connected to DevNet, navigate to the CC Scan Web UI at <http://scan.localhost:4000/>.

The default activity view shows the total CC balance and the Validator rewards.

The screenshot shows the main interface of the Canton Coin Scan Web UI. At the top left, it says "Explore, search and find answers to current network configuration details." At the top right, it indicates "The content on this page is computed as of round: 14". Below this, there are two main sections: "TOTAL CANTON COIN BALANCE" which displays "7,326,239.93606952 CC" and "36,631.1996803476 USD"; and "TOTAL APP & VALIDATOR REWARDS" which displays "20,520 CC" and "102.6 USD".

TOTAL CANTON COIN BALANCE	
7,326,239.93606952 CC	36,631.1996803476 USD

TOTAL APP & VALIDATOR REWARDS	
20,520 CC	102.6 USD

Select the Network Info menu to view SV identification.

SUPER VALIDATOR OPERATIONS

Information Validator Onboarding Canton Coin Price Delegate Election Governance

General DSO Info Canton Coin Info CometBFT Debug Info Domain Node Status

Super Validator Information

svUser	administrator
svPartyId	sv::1220e29d956452e... ⓘ

Active Super Validators

sv	sv::1220e29d956452e... ⓘ
----	--------------------------

Decentralized Synchronizer Operations

dsoLeaderPartyId	sv::1220e29d956452e... ⓘ
dsoPartyId	DSO::12200af1e2b814... ⓘ
dsoEpoch	0

The Validators menu shows that the local validator has been registered with the SV.

CANTON COIN SCAN		Col 2	Col 3	Col 4	Col 5	Col 6
CREATED AT	VALIDATOR	SPONSOR				
02/11/2025 08:06	sv::12201b7f2459db0f8...	sv::12201b7f2...	THIS SV			
02/11/2025 08:05	quickstart-jpmiller-1::12...	sv::12201b7f2...	THIS SV			
02/11/2025 08:05	quickstart-jpmiller-1::12...	sv::12201b7f2...	THIS SV			

Observability Dashboard

In a web browser, navigate to <http://localhost:3030/dashboards> to view the observability dashboards. Select “Quickstart - consolidated logs”.

The default view shows a running stream of all services.

This screenshot shows the Digital Asset Quickstart - consolidated logs dashboard. The left sidebar includes Home, Starred, Dashboards (selected), Explore, Alerting, Connections, and Administration. The main area has tabs for Services (selected), All, Levels, and Search. It also includes Trace ID and Enter variable value fields. The Logs section displays a continuous stream of log entries. A specific entry is highlighted:

```
> 2024-12-11 11:25:17.113 [participant] Here is our list of the 15 most expensive database queries for daml.db-storage.write.executor with total of 1.2 s:
  count= 167 mean= 6.94 ms total= 1.0 s com.digitalasset.canton.time.Clock$Queued$anonfun$run$1(Clock.scala:77)
  count= 48 mean= 48.66 ms total= 0.1 s com.digitalasset.canton.store.db.DbSequencedEventStore$anonfun$store$2(DbSequencedEventStore.scala:121)
  count= 165 mean= 0.58 ms total= 0.1 s <unknown>
  count= 2 mean= 0.87 ms total= 0.1 s com.digitalasset.canton.participant.store.db.DbInfoLightSubmissionStore$anonfun$submit$1(DbInfoLightSubmissionStore.scala:275)
  count= 2 mean= 0.83 ms total= 0.0 s com.digitalasset.canton.participant.store.db.DbInfoLightSubmissionStore$anonfun$submit$2(DbInfoLightSubmissionStore.scala:276)
```

Below this, another entry is shown:

```
> 2024-12-11 11:25:16.977 [validator_backend] Here is our list of the 15 most expensive database queries for daml.db-storage.general.executor with total of 0.2 s:
  count= 8 mean= 6.03 ms total= 0.0 s org.ldcentralizedtrust.splice.store.db.DomainValidatorsStore$anonfun$listExpiredFromPayloadExpiry$3(DbMultiDomainValidatorsStore.scala:288)
  count= 4 mean= 8.98 ms total= 0.0 s org.ldcentralizedtrust.splice.store.db.DomainValidatorsStore$anonfun$lookupValidatorLicenseWithOfffsets$1(DbValidatorStore.scala:297)
  count= 2 mean= 16.78 ms total= 0.0 s org.ldcentralizedtrust.splice.validator.store.db.DomainValidatorsStore$lookupInstallByParty(DbValidatorStore.scala:164)
  count= 4 mean= 6.86 ms total= 0.0 s org.ldcentralizedtrust.splice.store.db.DomainValidatorsStore$lookupInstallByParty(DbValidatorStore.scala:334)
  count= 1 mean= 16.78 ms total= 0.0 s org.ldcentralizedtrust.splice.store.db.DomainValidatorsStore$lookupInstallByParty(DbValidatorStore.scala:254)
  count= 5 mean= 4.76 ms total= 0.0 s org.ldcentralizedtrust.splice.store.db.DomainValidatorsStore$anonfun$listSignedContracts$3(DbMultiDomainValidatorsStore.scala:254)
  count= 2 mean= 4.42 ms total= 0.0 s org.ldcentralizedtrust.splice.validator.store.db.DomainValidatorsStore$anonfun$listSignedContracts$4(DbMultiDomainValidatorsStore.scala:226)
```

Other log entries include:

- > 2024-12-11 11:25:16.978 [validator_backend] Using the client-cache [valiuditn \$com(2024-12-11T17:34:21.810Z)] to load following issuing rounds: Vector(11208, 11208, 11208).
- > 2024-12-11 11:25:16.938 [sequencer] Here is our list of the 15 most expensive database queries for daml.db-storage.general.executor with total of 0.0 s:
 count= 167 mean= 6.07 ms total= 0.0 s com.digitalasset.canton.store.db.ObsoletePreReadStore\$prehead\$1(ObsoletePreReadStore.scala:46)
- > 2024-12-11 11:25:16.318 [x] Completed processing PeriodicTask for = 2024-12-11T17:25:16.297Z with outcome: Updated domain params to 1
- > 2024-12-11 11:25:16.317 [x] Request (tid:fdd25356fe7e5db9b591b451d5294e) com.digitalasset.canton.topology.admin.v30.TopologyManagerReadService/ListDomainParametersState to participant-v:5802: succeeded(OK)
- > 2024-12-11 11:25:16.317 [x] Request (tid:fdd25356fe7e5db9b591b451d5294e) com.digitalasset.canton.topology.admin.v30.TopologyManagerReadService/ListDomainParametersState to participant-v:5802: received a message
- > 2024-12-11 11:25:16.312 [x] Request (tid:fdd25356fe7e5db9b591b451d5294e) com.digitalasset.canton.topology.admin.v30.TopologyManagerReadService/ListDomainParametersState to participant-v:5802: sending request
- > 2024-12-11 11:25:16.311 [x] Request (tid:c98649174f298bf1f8eed1cc43708) com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/listConnectedDomains to participant-v:5802: succeeded(OK)
- > 2024-12-11 11:25:16.311 [x] Request (tid:c98649174f298bf1f8eed1cc43708) com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/listConnectedDomains to participant-v:5802: received a message
- > 2024-12-11 11:25:16.303 [x] Request (tid:c98649174f298bf1f8eed1cc43708) com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/listConnectedDomains to participant-v:5802: sending request
- > 2024-12-11 11:25:15.965 [validator_backend] Completed processing PeriodicTask for = 2024-12-11T17:25:15.933Z with outcome: Updated domain params to 1
- > 2024-12-11 11:25:15.968 [validator_backend] Request (tid:fbaef1b8a964b5723c3dc4d3fa741366f) com.digitalasset.canton.topology.admin.v30.TopologyManagerReadService/ListDomainParametersState to participant-v:5802: succeeded(OK)
- > 2024-12-11 11:25:15.968 [validator_backend] Request (tid:fbaef1b8a964b5723c3dc4d3fa741366f) com.digitalasset.canton.topology.admin.v30.TopologyManagerReadService/ListDomainParametersState to participant-v:5802: received a message
- > 2024-12-11 11:25:15.948 [validator_backend] Request (tid:d447cb28e28c692b722b7d83dfc3) com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/listConnectedDomains to participant-v:5802: sending request
- > 2024-12-11 11:25:15.948 Validator backend Request (tid:d447cb28e28c692b722b7d83dfc3) com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/listConnectedDomains to participant-v:5802: succeeded(OK)

Change the services filter from “All” to “participant” to view participant logs. Select any log entry to view its details.

This screenshot shows the Digital Asset Quickstart - consolidated logs dashboard with the services filter set to "participant". The main area has tabs for Services (selected), participant, Levels, All, and Search. It also includes Trace ID and Enter variable value fields. The Logs section displays a list of log entries. One entry is selected:

```
> 2024-12-11 11:26:19.742 [participant] Request com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/ListConfiguredDomains by /172.19.0.18:44912: completed
```

Details for this entry:

Fields	Value
all _timestamp	2024-12-11T17:26:19.742Z
all _version	1
all container_id	f54d920d96fcfd9244003a77ab88ce42b0becd2be66c72f3c81e819a1faef8e67
all container_name	/participant
all detected_level	debug
all fluent_tag	f54d920d96fc
all level	DEBUG
all level_value	10000
all logger_name	c.d.c.n.g.ApiRequestLogger:participant=participant
all message	Request com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/ListConfiguredDomains by /172.19.0.18:44912: completed
all service_name	participant
all source	stdout
all thread_name	canton-env-ec-278
all trace_id	5241b43733c65a4015518a2b92b26e50

Links for this entry:

- TraceID: 5241b43733c65a4015518a2b92b26e50
- Tempo

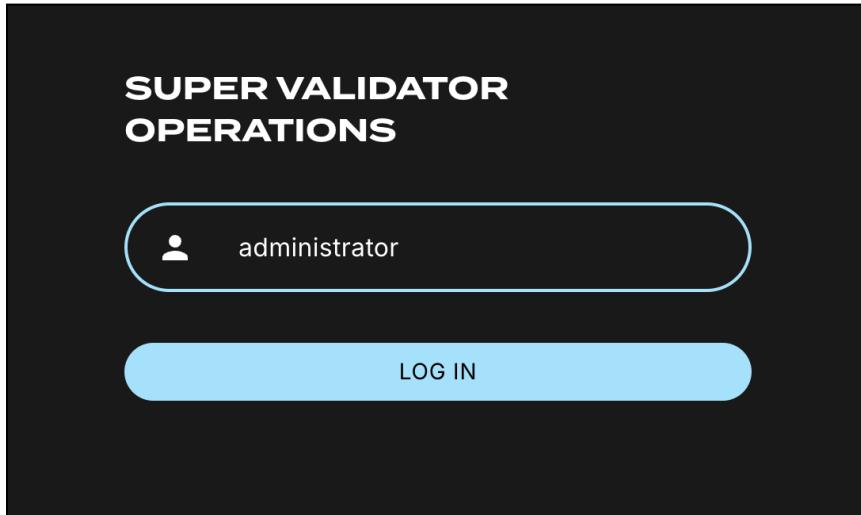
More log entries are listed below:

- > 2024-12-11 11:26:19.740 [participant] Request com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/ListConfiguredDomains by /172.19.0.18:44912: succeeded(OK)
- > 2024-12-11 11:26:19.740 [participant] Request com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/ListConfiguredDomains by /172.19.0.18:44912: sending response
- Request tid:5241b43733c65a4015518a2b92b26e50
- > 2024-12-11 11:26:19.739 [participant] Request com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/ListConfiguredDomains by /172.19.0.18:44912: received a message
- Request tid:5241b43733c65a4015518a2b92b26e50
- > 2024-12-11 11:26:19.735 [participant] Request com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/ListConfiguredDomains by /172.19.0.18:44912: succeeded(OK)
- > 2024-12-11 11:26:19.735 [participant] Request com.digitalasset.canton.admin.participant.v30.DomainConnectivityService/ListConfiguredDomains by /172.19.0.18:44912: completed

SV UIs

SV Nodes only run in LocalNet. As a quickstart app developer, you are not required to access the SV UI. However, if you're curious, you may explore the SV UI by navigating to <http://sv.localhost:4000/>. The SV view displays data directly from the validator.

Login as 'administrator'.



The UI shows information about the SV and lists the active SVs.

The Validator Onboarding menu allows for the creation of validator onboarding secrets.

The screenshot shows the Daml Super Validator Operations interface. At the top, there are navigation links: Information, Validator Onboarding (which is currently selected), Canton Coin Price, Delegate Election, Governance, and Logout. Below the navigation, the title "Validator Onboarding Secrets" is displayed. A blue button labeled "CREATE A VALIDATOR ONBOARDING SECRET" is visible. The main content area shows a table of Validator Licenses. The columns are "CREATED AT", "VALIDATOR", and "SPONSOR". There are three rows of data:

CREATED AT	VALIDATOR	SPONSOR
02/13/2025 15:13	quickstart-jpmiller-1::122077f1ae74e... 🔗	sv::122016716056ad8... 🔗 THIS SV
02/13/2025 15:13	quickstart-jpmiller-1::12203783f656... 🔗	sv::122016716056ad8... 🔗 THIS SV
02/13/2025 15:10	sv::122016716056ad8277a9220b78... 🔗	sv::122016716056ad8... 🔗 THIS SV

Development Journey in the CN QS Lifecycle

The CN QS provides a foundation for developing applications on the Global Synchronizer.

CN QS Components

The CN QS consists of three components that the developer may find of interest. These include development tools, LocalNet that simulates a Global Synchronizer on your laptop, and the sample application. Each component holds significance based on where the developer is in the lifecycle of the application.

Development Tools

The development tools in CN QS provide critical infrastructure that outlasts the sample application code. Understanding these tools informs decisions about which components to keep, modify, or replace as your application evolves.

Build System

The build system integrates Daml smart contract with the Java and TypeScript applications. Running `./gradlew build` generates code from the Daml model, packages contracts into DAR files, and prepares deployment.

To understand the project structure, dependencies, and root project configuration, examine `quickstart/build.gradle.kts`. For Daml-specific build configurations, review `quickstart/daml/build.gradle.kts`.

To extend the build system for your application, create parallel project structures in `quickstart/settings.gradle.kts`. These settings allow you to maintain your code alongside the original CN QS components while leveraging the same build infrastructure. Customize code generation by modifying the Gradle tasks in `quickstart/buildSrc/src/main/kotlin/` to target specific languages or adjust output formats.

As your application evolves, you can fine-tune dependency management across language boundaries, configure artifact publishing for CI/CD pipelines, and integrate with the Canton ledger APIs. The build system serves as the foundation that connects your Daml models to client applications.

When troubleshooting build issues, check the generated code in `build/generated-daml-bindings/` to verify that your Daml models are correctly translated to your target languages.

Understanding the build system can save extensive time in development efforts compared to creating custom build processes from scratch.

Makefile Command Interface

The Makefile provides standardized commands for common operations:

<code>make setup</code>	Configure environment variables and dependencies
<code>make build</code>	Build all components (Daml, backend, frontend)
<code>make start</code>	Start the application stack
<code>make console-app-provider</code>	Access Canton console for the provider
<code>make console-app-user</code>	Access Canton console for the user
<code>make shell</code>	Start Daml Shell for interactive testing

The Makefile serves as the primary control panel for interacting with the CN QS environment.

Run `make setup` to configure environment variables in `.env` files.

`make start` applies the appropriate environment settings and orchestrates all services through Docker Compose.

When you need direct access to the Canton ledger, use `make console-app-provider` to open an interactive console session.

`Makefile` integrates with Gradle to trigger builds and code generation with a single command, rather than needing to map complex Gradle tasks directly. Examine `makefile` to understand all available commands to streamline common development workflows and extend with your own custom commands as your application evolves.

Configuration Files

Modify the configuration files to match your application's requirements. Start with the Canton console configuration in `quickstart/config/canton-console/app.conf` to adjust ledger access permissions and admin operations. When you need to change network routing or add SSL certificates, edit the NGINX configurations in `quickstart/config/nginx/` directory.

Fine-tune your observability stack by modifying the configurations in `quickstart/config/o11y/` to capture application-specific metrics and create custom dashboards for monitoring your services. These files use standard formats ([HOCON](#) for Canton, [YAML](#) for Docker services, Grafana [JSON](#) for dashboards), making them easy to edit with standard tools.

Override configuration values by setting environment variables in your `.env` files rather than editing the configuration files directly. This approach makes it easier to incorporate upstream updates by keeping your customizations separate from the base configurations. For example, set `CANTON_ADMIN_PORT=5022` in your `.env` file to change the Canton admin API port without modifying the `app.conf` file.

When troubleshooting, examine these configuration files to understand how services are connected and what parameters control their behavior. As your application grows, create additional configuration files for your custom services following the same patterns established in the CN QS configurations.

Utility Tools

Leverage the CN QS utility tools during development and testing workflows. Use the build utilities in `quickstart/buildSrc/` to automate common development tasks. The

UnpackTarGzTask helps extract archive files while preserving permissions and symbolic links. The Java convention scripts standardize your application's build configuration across modules.

Configure your deployment environment by selecting the appropriate Docker Compose files in quickstart/docker/. Use compose-validator.yaml for validator nodes and adjust resource allocations with the resource-constraints-*.yaml files. Start the observability stack with docker-compose -f quickstart/docker/o11y/compose.yaml up to monitor your application's performance. The o11y directory integrates with Grafana dashboards defined in quickstart/config/o11y/ to provide real-time metrics visualization.

Examine these utilities early in your development process to understand their capabilities. Extend them to match your specific requirements rather than building similar functionality from scratch. For example, add custom test cases to the existing test framework or create new deployment scripts based on the provided templates.

We recommend keeping these utilities when you replace the sample application code. They provide infrastructure that would require significant effort to recreate. Copy them to your application's directory structure during the separation phase to maintain their functionality while decoupling from the original CN QS code.

LocalNet

LocalNet provides a self-contained Canton Network environment for development and testing. It includes all necessary components to simulate a Global Synchronizer on a single laptop without external dependencies.

Network Components

The LocalNet environment consists of three core components that work together to simulate a Canton Network. The Application Provider and User Validator nodes run Canton participant nodes to host your contracts and represent user participants. Each validator operates within its own preconfigured synchronizer.

The Global Synchronizer acts as the network coordinator through its Super Validator (SV). It runs a Canton synchronizer node that handles transaction ordering and conflict resolution using sequencer and mediator services. It verifies that all network participants maintain a consistent view of the distributed ledger.

A set of essential services supports these core components. PostgreSQL stores the ledger data, while Keycloak handles authentication and authorization. The Wallet Service manages digital assets and payments, and NGINX provides routing and SSL termination for secure communication between services.

Technical Implementation

The LocalNet environment is defined in the Docker Compose file:

- quickstart/compose.yaml

Key configuration files:

- quickstart/.env: Environment variables for the entire stack
- quickstart/docker/localnet.env: Network-specific configuration
- quickstart/config/canton-console/app.conf: Canton node configuration

LocalNet persists data through Docker volumes. Its network topology can be modified to meet specific business requirements. Canton console provides direct ledger access for debugging.

Access service logs in terminal using

```
make logs
```

Access git logs in terminal with

```
git log
```

Most teams maintain LocalNet throughout development, even after replacing the sample application. LocalNet provides a consistent testing platform that mirrors a production CN.

ScratchNet

ScratchNet is a term that refers to a LocalNet like deployment running on a single host that is accessible to more than one developer or automation. It is a middle ground between LocalNet and a decentralized DevNet. It's designed for scenarios requiring longer-running instances, more resources, CI/CD or integration testing activities, or multi-developer collaboration.

We've found that our clients prefer to set up a ScratchNet to create a more persistent LocalNet-like environment that can also be developed upon by a team.

Technical Implementation

A successful ScratchNet should include the following requirements:

- Server or VM (recommended minimum 64GB RAM, 16 CPU cores)
- Docker and Docker Compose
- External storage volumes for data persistence

- Network configuration that allows team access

Deployment Architecture

ScratchNet also requires persistent storage directories that are accessible across a team. Deploying ScratchNet architecture may use the following pattern:

```
# Clone CN QS repository to server
git clone https://github.com/digital-asset/cn-quickstart.git
cd cn-quickstart

# Create persistent storage directories
mkdir -p /mnt/scratchnet/postgres-data
mkdir -p /mnt/scratchnet/canton-data
```

Configure external volume mounts in a custom compose override file:

```
# scratchnet.yaml
version: '3.8'

services:
  postgres-splice-app-provider:
    volumes:
      -
      /mnt/scratchnet/postgres-data/app-provider:/var/lib/postgresql/data

  postgres-splice-app-user:
    volumes:
      -
      /mnt/scratchnet/postgres-data/app-user:/var/lib/postgresql/data

  postgres-splice-sv:
    volumes:
      -
      /mnt/scratchnet/postgres-data/sv:/var/lib/postgresql/data

  participant-app-provider:
    volumes:
      -
      /mnt/scratchnet/canton-data/app-provider:/canton-data

  participant-app-user:
    volumes:
      -
      /mnt/scratchnet/canton-data/app-user:/canton-data
```

Create a basic environment configuration.

```
# .env.scratchnet
# Unique network name
DOCKER_NETWORK=scratchnet

# External hostname where ScratchNet is accessible
EXTERNAL_HOSTNAME=scratchnet.example.com
Launch with persistent volumes:
# Set up environment
export ENV_FILE=.env.scratchnet

# Launch with volume persistence
COMPOSE_FILE=quickstart/compose.yaml:scratchnet.yaml make start
```

If your team is interested in setting up a ScratchNet environment, be sure to implement a regular, and preferably automated, backup strategy if you want to reuse or analyze generated data. Verify that access control is properly in place. We also suggest establishing a reliable way to monitor resource consumption, especially for extended runs. Your team may want to take advantage of resource management tools available through CN's Observability tools (Learn more in the Project Structure Guide), or you may choose to incorporate your own lightweight tools.

For example, a monitoring script in crontab can offer basic alerting.

```
#!/bin/bash
# db-monitor.sh - Run daily to monitor database growth

THRESHOLD=80
DB_PATH="/mnt/scratchnet/postgres-data"

USAGE=$(df -h $DB_PATH | grep -v Filesystem | awk '{ print $5 }' | sed 's/%//')
SIZE=$(du -sh $DB_PATH | awk '{ print $1 }')

echo "$(date): DB size is $SIZE, volume usage at $USAGE%" >>
/var/log/scratchnet-storage.log

if [ $USAGE -gt $THRESHOLD ]; then
    echo "ScratchNet PostgreSQL volume has reached ${USAGE}% capacity
($SIZE)"
fi
```

Containers can also be configured to automatically prune older data to reduce latency and maintain system integrity.

```
participant-app-provider:  
  environment:  
    CANTON_PARAMETERS:  
    "--canton.participants.participant.storage.write.pruning-interval=7d"
```

Sample Application

The CN QS includes a complete reference application that demonstrates Canton Network application patterns. While you'll likely replace this component entirely, understanding its architecture provides valuable insights for your own application design.

Application Components

Daml Models `quickstart/daml/licensing/`:

- Core business logic implemented as smart contracts
- License and AppInstall templates demonstrate multi-party workflows
- Integration with Splice

Backend Service `quickstart/backend/`

- Java Spring Boot application
- Ledger API integration for contract creation and exercise
- REST API exposing contract operations to frontend
- Automated code generation from Daml models

Frontend `quickstart/frontend/`

- React/TypeScript single-page application
- Component-based architecture with state management using React hooks
- REST API integration with backend service

Technical Implementation

The API Design is defined in `quickstart/common/openapi.yaml`. It contains the RESTful API definitions, establishes the JSON schema for request/response objects, provides error handling conventions, and creates authentication patterns.

Development Lifecycle

We've observed five distinct phases of the CN QS development journey. Each phase presents unique strategies for interacting with the CN QS.

Learning Phase

(½ - 2 weeks)

Often the first interaction with the CN QS is understanding how to get the environment running. The next goal is to explore the application and develop knowledge around the architecture and its workflow. It's also important to learn how to navigate the most common observability dashboards and move between LocalNet and DevNet.

The most direct update strategy in this phase is to regularly update your local copy of the CN QS by making a `git pull` from the main branch.

```
# Initial setup
git clone https://github.com/digital-asset/cn-quickstart.git
cd cn-quickstart

# Regular updates during learning
git pull origin main

# Environment customization (only if needed)
echo 'export PARTY_HINT="company-name"' > .envrc.private
direnv allow
```

Experimentation Phase

(1-2 weeks)

In this phase, you'll reinforce your understanding of the CN QS by experimenting with the configurations, exploring the Ledger and CN app APIs, and modify the Daml code, Java client, and Makefile to test integration patterns.

At this phase, you may want to establish upstream tracking to selectively incorporate changes.

```
# Set up upstream tracking
git remote add upstream
https://github.com/digital-asset/cn-quickstart.git

# Create a branch for experiments
git checkout -b experiments

# Periodically incorporate upstream changes
git fetch upstream
git merge upstream/main
```

Development Phase

(2-3 weeks)

This is where you begin building your own application alongside the CN QS sample application. Many developers create their new app in parallel code directories to the CN QS application to learn from the CN QS while building their own application.

```
cn-quickstart/
└── quickstart/          # Original CN QS code
    ├── daml/             # CN QS Daml code
    ├── backend/           # CN QS backend service
    └── frontend/          # CN QS frontend

└── myapp/               # Your application code
    ├── daml/              # Your Daml models
    ├── backend/            # Your backend services
    └── frontend/           # Your frontend code
```

Developers may generate new Daml packages, new client code in languages other than Java or TypeScript, UI elements, CI/CD integration, and unit tests.

Gradle Settings

When you develop parallel directories, remember to update your build configuration to include both structures.

```
// In settings.gradle.kts
include("quickstart:daml")
include("quickstart:backend")
include("quickstart:frontend")

include("myapp:daml")
include("myapp:backend")
include("myapp:frontend")
```

Maintain separate build files for application components.

```
// In myapp/backend/build.gradle.kts
dependencies {
    // Reference CN QS components if needed
    implementation(project(":quickstart:daml"))

    // Your specific dependencies
```

```
    implementation("your.dependency:library:1.0.0")
}
```

Environment Variables

Use `.envrc.private` for local overrides.

```
# Override CN QS defaults
export PARTY_HINT="your-company"
export DAML_SDK_VERSION="your-version"

# Add your application-specific variables
export MY_APP_CONFIG="/path/to/config"
```

Create separate environment files for your application.

```
# In myapp/.env
MY_APP_PORT=8080
MY_APP_DB_URL=jdbc:postgresql://localhost:5432/myapp
```

Docker Compose

Create custom compose files that extend the CN QS configuration.

```
# In myapp/compose.yaml
version: '3.8'

# Import the CN QS services
include:
  - ./quickstart/compose.yaml

# Add your services
services:
  myapp-backend:
    build: ./backend
    depends_on:
      - postgres
      - participant
    environment:
      - DB_URL=${MY_APP_DB_URL}
```

Use profiles to selectively enable groups of services.

```
# Start with CN QS and your services
docker-compose --profile quickstart --profile myapp up

# Start only your services (once they are able to run independently)
docker-compose --profile myapp up
```

Separation Phase

Over the course of a few weeks, CN developers have gained enough experience and their new application's complexity begins to exceed that of the CN QS. At this point, the CN QS is no longer helpful and the developer is advised to cut ties with the sample application.

To remove dependence on the CN QS, delete the example application directories, adjust gradle files, change the environment variable files, and remove the upstream connection in git.

The developer's source code repository is disconnected from the CN QS repository. It's advisable to write a bridge document that maps application components to their origins in the CN QS to create a historical development record.

```
# Remove the CN QS remote
git remote remove upstream

# Clean up unused directories (after backing up if needed)
rm -rf quickstart/

# Update build files to remove CN QS references
# Edit settings.gradle.kts, build.gradle.kts, etc.
```

Ongoing Updates

By now, your application is likely to outgrow the capabilities of the CN QS. However, you may want the ability to update the development tooling or LocalNet support. The CN QS continuously adds more tooling features and updates existing tool versions.

This process includes periodically checking into CN QS, reviewing the ChangeLog to see what is new, and then selecting components you'd like to include in your application. You'll find the CN QS to be a source for improvements, rather than a direct dependency.

We recommend establishing a regular schedule (monthly or quarterly) to review CN QS updates.

Your update strategy may include creating a temporary clone of the CN QS to review changes, manually incorporating them into your project, and then removing the temporary clone.

```
# Temporary clone to review changes
git clone https://github.com/digital-asset/cn-quickstart.git
cn-quickstart-temp
cd cn-quickstart-temp
git log --since="3 months ago" --pretty=format:"%h - %an, %ar : %s"

# After identifying useful changes, manually incorporate them into
your project
# Then remove the temporary clone
cd ..
rm -rf cn-quickstart-temp
```

Every development team's journey is unique. Adapt these strategies to fit your specific needs, team structure, and application requirements. As a CN developer, your goal is to find an approach that supports your development goals while also using the CN QS as a foundation to accelerate your development lifecycle.

Upgrades On The Global Synchronizer

The SVs periodically implement upgrades to the Global Synchronizer to improve functionality, resolve issues, and introduce new features. As a node operator or application provider you should be aware of the three types of upgrades that may occur.

Type 1: Backward-Compatible Changes

Type 1 upgrades involve backward-compatible changes to the Splice applications and/or modifications to the behavior of the Canton synchronization layer. These non-breaking changes occur on Mondays, every week.

While validators can operate effectively when behind by a Splice version or two, the SVs recommend keeping your node up to date with weekly upgrades. It's worth noting that "skip upgrades" (jumping multiple versions at once) are not officially tested by the SVs, so while they generally work, they come with increased risk.

Type 2: Daml Model Changes

Type 2 upgrades modify the Daml models that underlie the Splice applications. These changes introduce a fork in the application chains and occur every few months.

The process for Type 2 upgrades begins with distribution of the new Daml models through Type 1 upgrades, followed by an offline Canton Improvement Proposal (CIP) that must be approved by the SV node owners. Next, the SVs conduct an onchain vote to establish a specific date and time when the new models take effect. At this cutoff point, only validators running the most recent Splice version are able to participate in transactions using the new models. Validators that haven't adopted the latest version are unable to participate in transactions.

Type 3: Non-Compatible Protocol Changes

Type 3 upgrades involve fundamental changes to the Canton synchronization protocol. These major upgrades require downtime (sometimes called Hard Migrations) and occur every three to four months.

The implementation of Type 3 upgrades requires a Canton Improvement Proposal (CIP) approved through an offchain vote, followed by an onchain vote by the SVs to schedule the upgrade. These migrations impact all SVs and Validators, requiring a coordinated transition from the prior protocol to the new one. Currently, Canton requires all nodes to migrate together during these upgrades.

Preparing for Upgrades

Application providers should maintain nodes on DevNet, TestNet, and MainNet to guarantee smooth operations during upgrades. By maintaining nodes across all three environments you substantially increase the likelihood that MainNet upgrades proceed without disrupting your services or customers.

Keycloak in the CN QS

Keycloak is an open-source Identity and Access Management (IAM) solution that provides authentication, authorization, and user management for modern applications and services. It acts as a centralized authentication server that handles user logins, session management, and security token issuance.

The CN QS uses Keycloak to provide secure authentication across its distributed architecture. Keycloak maintains separation between authentication concerns and business logic.

Realm Structure

The CN QS defines two Keycloak realms. The AppProvider realm manages authentication for services and users on the provider side of the application. The AppUser realm handles

authentication for the consumer side. When components like validators or participant nodes receive requests, they validate the authentication tokens against the appropriate realm.

Keycloak Configuration

The default .env configuration includes predefined users in each realm:

- **User "Pat"** (AUTH_APP_PROVIDER_WALLET_ADMIN_USER_NAME=pat)
- **UUID:** 553c6754-8879-41c9-ae80-b302f5af92c9
(AUTH_APP_PROVIDER_WALLET_ADMIN_USER_ID)

AppUser Realm:

- **User "Alice"** (AUTH_APP_USER_WALLET_ADMIN_USER_NAME=alice)
- **UUID:** 92a520cb-2f09-4e55-b465-d178c6cfe5e4
(AUTH_APP_USER_WALLET_ADMIN_USER_ID)
- **Password:** abc123 (AUTH_APP_USER_WALLET_ADMIN_USER_PASSWORD)

Customizing Keycloak for Business Needs

You can customize the Keycloak configuration to meet your specific business requirements.

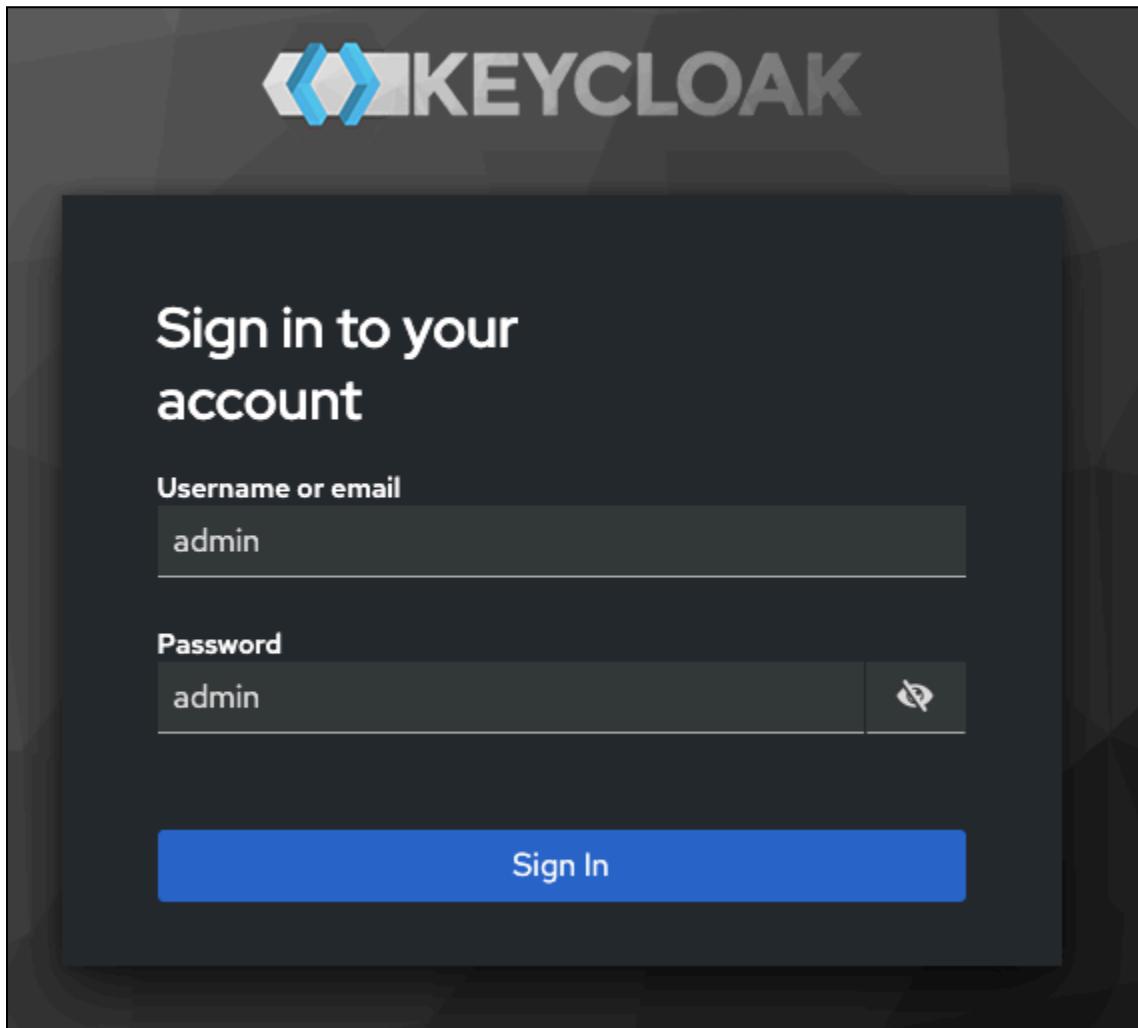
Accessing the Admin Console

The Keycloak Admin Console is available at:

```
http://keycloak.localhost:8082/admin/master/console/#/master
```

To log in use the default credentials:

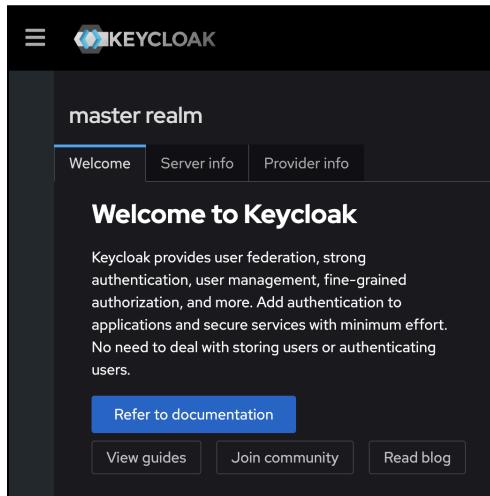
- **Username:** `admin`
- **Password:** `admin`



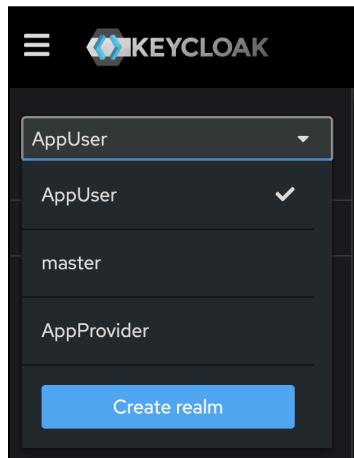
Customization Scenarios

Add a New User

1. Log in to the Keycloak Admin console

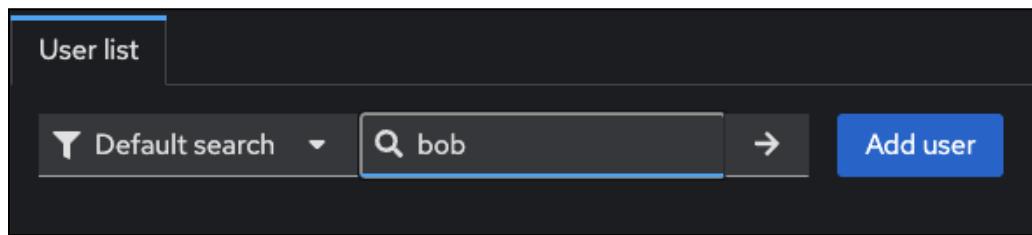


2. Select the appropriate realm (AppProvider or AppUser)



3. Navigate to the “Users” -> “Add user”

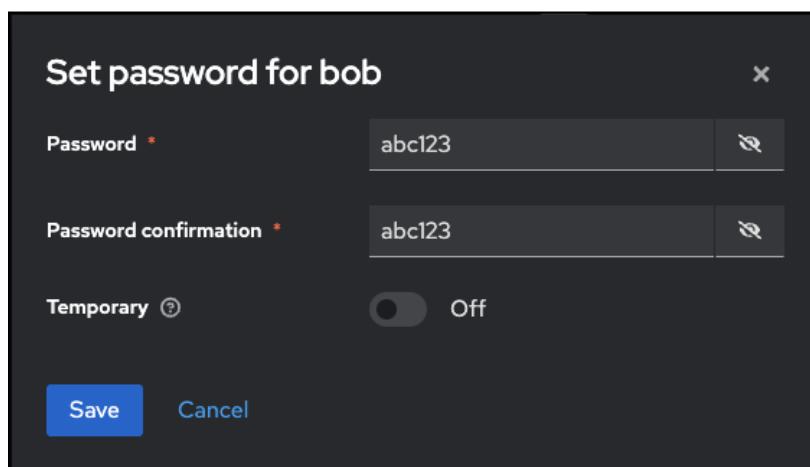
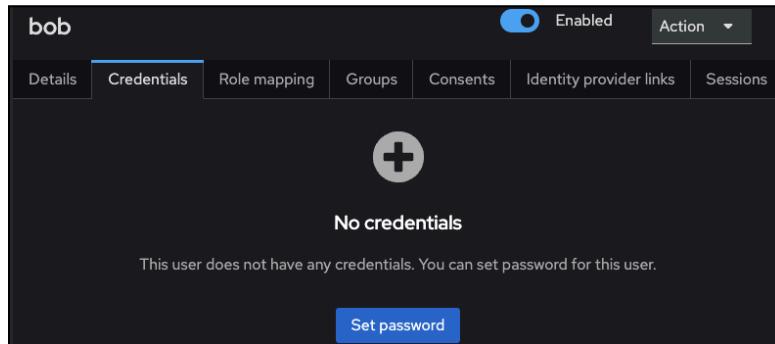
The screenshot shows the 'Users' page in Keycloak. On the left, there is a sidebar with navigation links: 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users' (which is selected and highlighted in blue), 'Groups', 'Sessions', and 'Events'. The main content area has a title 'Users' and a sub-section 'User list'. It includes a search bar ('Default search' and 'Search user') and a blue 'Add user' button. A table below lists two users: 'adrian' and 'alice'. The table columns are 'Username', 'Email', 'Last name', and 'First name'. Each user row has a delete icon and a more options icon. At the bottom of the table are pagination controls '1 - 2' and navigation arrows.



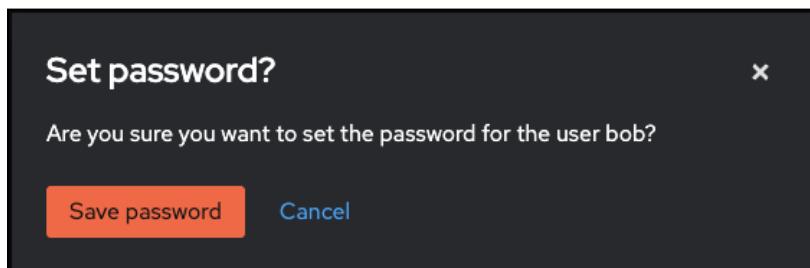
4. Fill in the user details and click **Create**

A screenshot of a user creation form titled 'General'. The form includes fields for 'Username *' (containing 'bob'), 'Email' (containing 'Email'), 'First name' (containing 'First name'), 'Last name' (containing 'Last name'), and 'Groups' (with a 'Join Groups' button). At the bottom are 'Create' and 'Cancel' buttons.

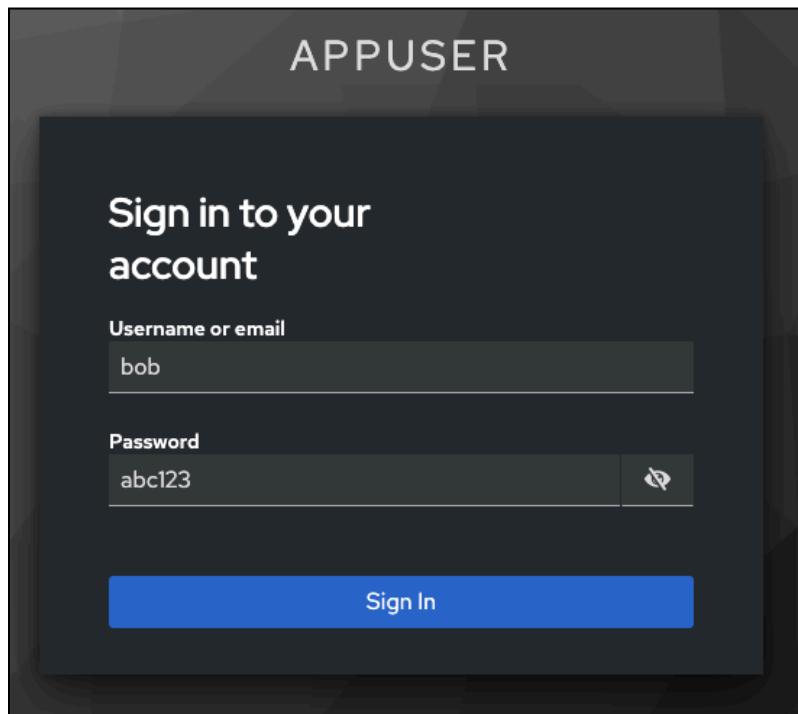
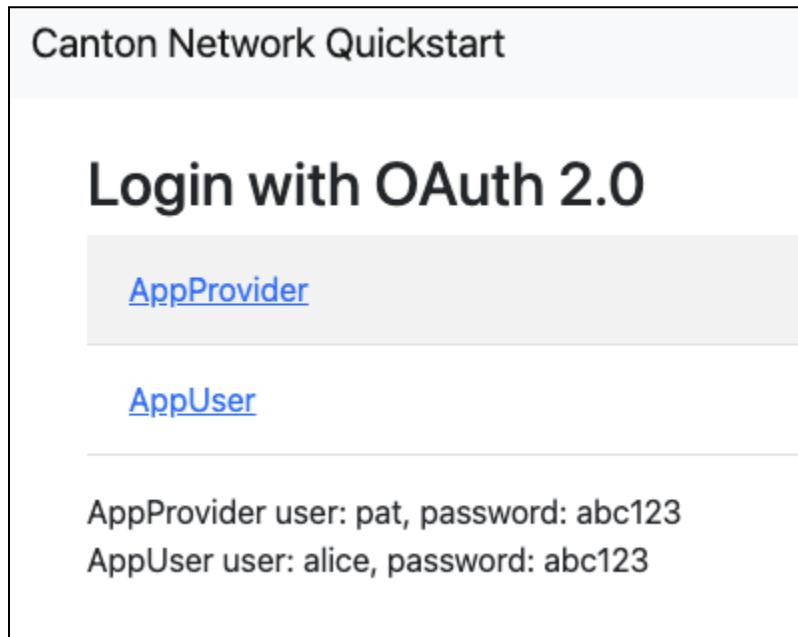
5. Go to the **Credentials** tab to set a password



6. Save the password



7. You can now sign in using the new user and their password.
 - a. Click **AppUser**



8. Bob is now a user

The screenshot shows the "App Installs" section of the application. At the top, there is a note: "Note: Run `make create-app-install-request` to submit an AppInstallRequest".

Modify Client Settings

1. Select the appropriate realm
2. Navigate to **Clients** -> Select the client to modify

The screenshot shows the Keycloak admin interface for managing clients. On the left, a sidebar menu is open under the 'AppUser' realm. The 'Clients' option is selected, highlighted in blue. The main content area is titled 'Clients' with a subtitle: 'Clients are applications and services that can request authentication of a user.' Below this is a navigation bar with tabs: 'Clients list' (which is active), 'Initial access token', and 'Client registration'. There is also a search bar labeled 'Search for client' with a magnifying glass icon, a 'Create client' button in blue, and an 'Import client' link. A 'Refresh' button is also present. At the bottom right, there is a pagination indicator '1 - 10' with arrows. The main table lists ten clients:

Client ID	Name	Type	Description	Home URL
accounting	client_1	OpenID Connect	-	http://keycloak.localhost:8082/realm/AppUser/account/
accounting	client_2	OpenID Connect	-	http://keycloak.localhost:8082/realm/AppUser/account/
admin	client_3	OpenID Connect	-	-
app	client_4	OpenID Connect	-	-
app	client_5	OpenID Connect	-	-
app	client_6	OpenID Connect	-	-
app	client_7	OpenID Connect	-	-
broker	client_8	OpenID Connect	-	-
realm	client_r	OpenID Connect	-	-
security	client_s	OpenID Connect	-	http://keycloak.localhost:8082/admin/AppUser/console/

3. Update settings per your needs

The screenshot shows the 'Clients' section of the Digital Asset management interface. A client named 'app-user-wallet' is selected, which is an 'OpenID Connect' type. The client is currently 'Enabled'. The 'General settings' tab is active, displaying fields for Client ID (app-user-wallet), Name (client_app-user-wallet), and Description (empty). An 'Always display in UI' toggle is set to 'Off'. The 'Access settings' tab is also visible, showing empty fields for Root URL and Home URL, and a Valid redirect URIs field containing 'http://wallet.localhost:2000'. A sidebar on the right lists other configuration sections: General settings, Access settings, Capability config, Login settings, and Logout settings. At the bottom are 'Save' and 'Revert' buttons.

4. Save changes

Add a New Client

1. Select the appropriate realm
2. Navigate to “Clients” -> “Create”

The screenshot shows the 'Clients list' page. It features three tabs at the top: 'Clients list' (selected), 'Initial access token', and 'Client registration'. Below the tabs is a search bar with a magnifying glass icon and the placeholder 'Search for client'. To the right of the search bar is a large blue 'Create client' button.

3. Configure the client's general settings. Click **Next** for additional configuration options

The screenshot shows the 'Create client' page with a dark theme. On the left, a sidebar lists three steps: 1. General settings (selected), 2. Capability config, and 3. Login settings. The main area contains fields for 'Client type' (set to 'OpenID Connect'), 'Client ID' (a redacted input field), 'Name' (a redacted input field), and 'Description' (a redacted input field). A toggle switch for 'Always display in UI' is set to 'Off'. At the bottom, there are 'Back', 'Next' (highlighted in blue), and 'Cancel' buttons.

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings 2 Capability config 3 Login settings

Client type ⓘ OpenID Connect

Client ID * ⓘ

Name ⓘ

Description ⓘ

Always display in UI ⓘ Off

Back Next Cancel

4. Configure additional settings

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings 2 Capability config 3 Login settings

Client authentication: Off

Authorization: Off

Authentication flow:

- Standard flow
- Direct access grants
- Implicit flow
- Service accounts roles
- OAuth 2.0 Device Authorization Grant
- OIDC CIBA Grant

Back Next Cancel

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings 2 Capability config 3 Login settings

Root URL

Home URL

Valid redirect URIs

Valid post logout redirect URIs

Web origins

Add valid redirect URIs

Add valid post logout redirect URIs

Add web origins

Back Save Cancel

5. Save the client

Update Environment Variables

After making changes to Keycloak configuration, you may need to update the corresponding environment variables in the `.env` file:

1. The Keycloak user must have the same ID as the ledger user's ID. This is not the party id.
 2. For client changes, update the corresponding client ID and secret
 3. For user changes, update the corresponding user ID and credentials
 4. Restart the services to apply the changes:

make stop && make start

Troubleshooting

Login Failures:

- ## 1. Verify Keycloak is running: `make status`

Find **keycloak** near **grafana** and **loki** in the list.

Keycloak should show as “healthy”

grafana	grafana/grafana:11.1.5	"/run.sh"	grafana	30 minutes ago	Up 30 minutes	0.0.0.0:3000->3000/tcp
keycloak	quay.io/keycloak/keycloak:26.1.0	"/opt/keycloak/bin/w...	keycloak	30 minutes ago	Up 30 minutes	0.0.0.0:8080->8443/tcp, 9000/tcp
keycloak-postgres	postgres:14	"docker-entrypoint.s...	keycloak-postgres	30 minutes ago	Up 30 minutes	5432/tcp
loki	grafana/loki:3.1.1	"/usr/bin/loki -con...	loki	30 minutes ago	Up 30 minutes	3100/tcp

- ## 2. Check keycloak credentials in .env file

```
AUTH_APP_USER_ISSUER_URL_BACKEND=http://nginx-keycloak:8082/realms/AppUser
# for backend
AUTH_APP_USER_ISSUER_URL=http://keycloak.localhost:8082/realms/AppUser
# for backend, wallet-ui
AUTH_APP_PROVIDER_ISSUER_URL=http://keycloak.localhost:8082/realms/AppProvider
# for backend oidc client conf, wallet-ui
AUTH_APP_PROVIDER_ISSUER_URL_BACKEND=http://nginx-keycloak:8082/realms/AppProvider
# for backends
```

3. Check that the Keycloak user ID matches the ledger user ID

a. App User

- i. Compare the **ID** value in Keycloak's User Details with the

AUTH_APP_USER_WALLET_ADMIN_USER_ID value in .env.

AUTH_APP_USER_WALLET_ADMIN_USER_ID=92a520cb-2f09-4e55-b465-d178c6cf5e4

The screenshot shows the Keycloak User details page for a user named 'alice'. The user is enabled. The 'Details' tab is selected, showing the following information:

Attribute	Value
ID *	92a520cb-2f09-4e55-b465-d178c6cf5e4
Created at *	2/18/2025, 4:33:20 PM

Below the table, there is a 'Required user actions' section with a 'Select action' button.

b. App Provider

Compare the **ID** value in Keycloak's User Details with the

AUTH_APP_PROVIDER_WALLET_ADMIN_USER_ID value in .env.

AUTH_APP_PROVIDER_WALLET_ADMIN_USER_ID=553c6754-8879-41c9-ae80-b302f5af92c9

The screenshot shows the Keycloak User details page for a user named 'pat'. The user is enabled. The 'Details' tab is selected, showing the following information:

Attribute	Value
ID *	553c6754-8879-41c9-ae80-b302f5af92c9
Created at *	2/5/2025, 11:19:20 PM

Below the table, there is a 'Required user actions' section with a 'Select action' button.

Learn more about using Keycloak through their documentation portal:

[Keycloak Official Documentation](#)

[Keycloak Server Administration Guide](#)

[Securing Applications with Keycloak](#)

Next Steps

You've completed a business operation in the CN QS and have been introduced to the basics of the Canton Console, Daml Shell, the SV UIs, the Global Synchronizer, and Keycloak.

Learn more about Daml Shell and the project structure in the Project Structure Guide.