



# Observability & Troubleshooting (Overview)

Colony Network Quickstart Guide | 2025

Version: 0.9.0-2025-02-20

**Note:** The screenshots in this guide are currently taken from multiple sessions over multiple days and therefore are inconsistent with each other (and in some places the text). This will be rectified once some of the updates to Quickstart currently in flight are committed.

## Contents

### [Overview of Observability](#)

[The LocalNet Configuration](#)

[Observability Overview](#)

[Daml Shell](#)

[Grafana](#)

[Direct Postgres Access](#)

[Interactive Debugger](#)

### [Observability and Tracing](#)

[Correlation Identifiers](#)

[Direct Ledger Inspection using Correlation Identifiers](#)

[Correlated Logs and Traces using Correlation Identifiers](#)

# Overview of Observability

The Canton Network Quickstart deployment configuration includes a full observability suite. Tools preconfigured for monitoring and troubleshooting distributed Canton applications—both in development and production. The observability suite provides three key types of monitoring data:

- **consolidated structured logs** for application and system events
- **distributed traces** that visualize end-to-end transaction flows; and
- **metrics** for monitoring key performance indicators.

The suite allows data types to be correlated with each other to provide insights for root cause analysis. In addition, the Canton Ledger also provides a variety of correlation and tracing ids that permit tracking transaction provenance across multiple organizations and environments.

## The LocalNet Configuration

The Quickstart runtime configuration is defined in `.env.local`, which allows each developer to switch between running a LocalNet or DevNet application deployment; and, whether or not to bring up a local deployment of the Observability Stack. This file can be created using `$ make setup`, which wraps the command `$ ./gradlew configureProfiles --no-daemon --console=plain --quiet`, or can be edited manually to set environment variables `LOCALNET_ENABLED` and `OBSERVABILITY_ENABLED` to `true` or `false` as desired.

The LocalNet runtime configuration is handled by `docker-compose` configured in `compose.yaml` using environment variables from `.env` in the `quickstart/` project root directory. As a result the usual Docker commands and tooling applies.

Immediately useful commands you probably already know:

- `$ docker ps` lists the running containers.
- `$ docker logs [-f] <container>` fetches the logs of a container, and follow the logs with the `-f` option
  - If the system is not working well to the extent you do not trust the observability stack (discussed later), `docker logs backend-service` is a good place to start looking for errors that might provide an insight into what has gone wrong
- `$ docker restart <container>` for those instances where a container seems to have become stuck

## Observability Overview

The Quickstart application has been built to provide the foundation for a production Daml application. As such it includes a full observability configuration which is helpful to troubleshoot or

debug an application when running on the LocalNet. In order to provide a working demo Quickstart has naturally had to be opinionated regarding the choice of technologies, selecting from modern commonly used technologies. The platform itself is agnostic, and individual components can and should be replaced as required by your team.

The current troubleshooting and debugging services include:

- Local ledger inspection via Daml shell (<https://docs.daml.com/tools/daml-shell/index.html>)
- Datasource collection and management via **OpenTelemetry**
  - This uses the **OTEL Collector** (<https://opentelemetry.io/docs/collector>)
- Metrics are aggregated using **Prometheus** (<https://prometheus.io/>)
- Logs are consolidated using **Loki** (<https://grafana.com/oss/loki/>)
- Traces are aggregated using **Tempo** (<https://grafana.com/oss/tempo/>)
- Aggregated observations (metrics, logs, and traces) are viewable via **Grafana** (<https://grafana.com/oss/grafana/>) which acts to allow hyperlinked exploration of the Observability fields.

## Daml Shell

Daml Shell is a terminal application that provides interactive local ledger inspection on top of PQS. Quickstart is configured to launch Daml Shell in a docker container and is configured to connect to the included application provider's PQS instance. This is easiest to access via the toplevel project scripts accessed via **make** from **quickstart/**. To see this in action, build and start the quickstart app then:

Run **\$ make create-app-install-request** to use **curl** to submit the **create AppInstallRequest ...** command to the ledger<sup>1</sup> to initiate user onboarding<sup>2</sup>. Then you can use the following Daml Shell commands:

- > **active** to see a summary of the contracts you created; and,
- > **active quickstart-licensing:Licensing.AppInstall:AppInstallRequest** to see the contract details for any Asset contracts on the ledger; finally,
- > **contract [contract-id from the previous command]**<sup>3</sup> to see the full detail of the **AppInstallRequest** contract on the ledger
- > **help [command]** provides context help for daml shell commands<sup>4</sup>

---

<sup>1</sup> Specifically this sends a **CreateCommand** to the **submit-and-wait** service on the Application User's participant node.

<sup>2</sup> See the Canton Network Quickstart Guide "Project Structure" for more details on this

<sup>3</sup> Daml shell has tab completion on most command arguments, including the Template Id argument to **active** and the Contract Id argument to **contract**.

<sup>4</sup> Further documentation is available at <https://docs.daml.com/tools/daml-shell/index.html>

```
[12:34:29] ~/src/da/cn-quickstart/quickstart {main =>
✓ % make shell
docker compose -f docker/daml-shell/compose.yaml --env-file .env run --rm daml-shell || true
Connecting to jdbc:postgresql://postgres-splice-app-provider:5432/scribe...
Connected to jdbc:postgresql://postgres-splice-app-provider:5432/scribe
postgres-splice-app-provider:5432/scribe> active
Identifier                                         Type   Count
quickstart-licensing:Licensing.AppInstall:AppInstallRequest    Template 1
splice-amulet:Splice.Amulet:ValidatorRight      Template 1
splice-wallet:Splice.Wallet.Install:WalletAppInstall    Template 1
postgres-splice-app-provider:5432/scribe 40 → 4a> active quickstart-licensing:Licensing.App
Install:AppInstallRequest
Create at   Contract ID  Contract Key          Payload
4a         00201e099af26622c9ff...          dso: DS0::1220b1e6439dbc1cb181464db0ae1d263ed5af3568ae113f7
          ↳ 276879a60e600b922b7
          meta:
          ↳ values:
          ↳ user: Org1::1220fb5b633584568487994789de782ce13ced8939ef1a5
          ↳ d069b4ce245e6771d60dc
          provider: AppProvider::1220f41ccb6ad2a4e314e8d3cf9b0a6efb28
          ↳ 02274343d8cc3e1ea8d6a5ef1dcbb5
postgres-splice-app-provider:5432/scribe 40 → 4a> contract 00201e099af26622c9ff2a017238c626
7d918a15968adb167d13638b888d9460cdca10122037ee85f1a369a1070386397540246cd703bcfd713b680734
6607970e38ced94
Identifier                                         quickstart-licensing:Licensing.AppInstall:AppInstallRequest
Type   Template
Created at 4a (not yet active)
Archived at <active>
Contract ID 00201e099af26622c9ff...
Event ID #1220e117419dace8f88b...:0
Contract Key
Payload          dso: DS0::1220b1e6439dbc1cb181464db0ae1d263ed5af3568ae113f7276879a60e600b9
          ↳ 22b7
          meta:
          ↳ values:
          ↳ user: Org1::1220fb5b633584568487994789de782ce13ced8939ef1a5d069b4ce245e677
          ↳ 1d60dc
          provider: AppProvider::1220f41ccb6ad2a4e314e8d3cf9b0a6efb2802274343d8cc3e1
          ↳ ea8d6a5ef1dcbb5
postgres-splice-app-provider:5432/scribe 40 → 4a>
```

## Grafana

The Grafana interface is accessible via its web interface which is port-mapped to <http://localhost:3030/>, and can be opened in the current browser from the command line using `make open-observe`.

It is recommended that the focus of your debugging should be on using the trace and log facilities provided by Grafana and ledger inspection using Daml Shell. Ensuring that the exported logs and traces are sufficient to support debugging during development also provides assurance that they will be sufficient to support diagnostics in production.

There is additional access configured into the quickstart that can assist with debugging on LocalNet. To reiterate, best practice is to use the same diagnostic tools for development as you will for production. If you add a log line that then allows you to identify and fix a bug in development, then keeping it around at `trace` or `debug` log levels increases your operational readiness. Conversely, in one sense, using a tool that won't be available in production to debug in development reduces your operational readiness.

## Direct Postgres Access

All persistent state in the example application is stored in one or more postgres databases. You can use the postgres configuration in `.env` to connect directly to these instances.

```
$ docker exec -it <postgres container> psql -v --username <.env username> --dbname <.env dbname> --password
```

For example: if you connect to the `postgres-splice-app-provider` container (default username `cnadmin`, dbname `scribe`, and password `supersafe`; then you can use the SQL interface to PQS to examine the app-provider's participant's local ledger. The SQL API to PQS is documented in the daml documentation (<https://docs.daml.com/query/pqs-user-guide.html#>).

## Interactive Debugger

If you review the `compose.yaml` file and examine the configuration for backend-service you will see the lines:

```
backend-service:
  environment:
    ...
    JAVA_TOOL_OPTIONS: "-javaagent:/otel-agent.jar
    -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005"

  ports:
    - "${BACKEND_PORT}:8080"
    - "5055:5005"
```

This enables remote debugging of the java component backend in the user application (backend-service). You can use this to connect an IDE Debugger to the service at runtime if required. Keep in mind that we recommend your first resort be Grafana and the consolidated logs in Loki, as this ensures the system remains debugable in production.

# Observability and Tracing

Faulty distributed systems can be notoriously hard to diagnose. Quickstart provides, at the start of a project, the sort of observability and diagnostics facilities often only developed toward the end of the project. Simplifying diagnostics for new Canton Network Applications from the outset of each project is one of the motivations behind the development of Quickstart.

The links in the overview include the official user and reference documentation for the various tools included in Quickstart. While there is no substitute for the official documentation, it is hoped the following tour of the capabilities configured into Quickstart can provide a starting point for your own experimentation.

## Correlation Identifiers

Inspecting any distributed system invariably begins by correlating identifiers—Canton is no different in that regard. Canton can accept and/or generate a number of identifiers suitable for correlating across both time, various nodes, and the evolving state of the ledger.

A few of the key identifiers to be aware of are:

Useful Correlation Identifiers		
Identifier	Specified by	Scope
ApplicationId	The Ledger Client	Identifies the ledger client during command submission and processing.
WorkflowId	The Ledger Client	Identifies the business process. Persisted to the ledger
CommandId	The Ledger Client	Identifies the business “act” associated with a ledger command. Persisted to the ledger. Visible only to the submitting party. Common across retries.
SubmissionId	The Ledger Client	Identifies an individual ledger submission to a participant node.
TransactionId	Daml Ledger	Global identifier for a committed transaction to the ledger. Only visible to participant nodes that witness or are informed of the transaction <sup>5</sup> .

<sup>5</sup> A key differentiator of Canton from all other level one blockchains is that it offers privacy. It does this by enforcing right-to-know, rather than via secrecy-via-obscurity and/or via pseudo-anonymity. Canton provides two privacy guarantees: Even in encrypted form (sub-)transactions are only transmitted to participant nodes with a right to be informed of them; and, participant nodes will be informed of every (sub-)transaction they have a right to be informed of. For details on how Canton defines “right” and other aspects of this see the Daml Ledger Privacy Model (<https://docs.daml.com/concepts/ledger-model/ledger-privacy.html#privacy>)

### Useful Correlation Identifiers

Identifier	Specified by	Scope
Ledger EventId	Daml Ledger	Global identifier for a node within a committed transaction tree corresponding to a ledger event.
Trace/Span Id <sup>6</sup>	Ledger Client (or upstream)	Accepted by GRPC/HTTP ledger interfaces and honoured throughout the Canton Network code. Where one is not provided may sometimes be generated internally to provide tracing support within the network.
Ledger Offset	Participant Node	The height of a transaction within the local linearization of the ledger by a participant node <sup>7</sup> .
ContractId	Daml Ledger	Global identifier for a contract that was created successfully on the ledger at some point. If the contract has been subsequently archived the id remains a stable and valid way to refer to it even though the associated contract can no longer be used.
TemplatedId	Daml Application	Combined with a Packageld this provides a global identifier for a Daml smart contract.
Party Id	Participant Node	Global, potentially non-unique, identifier for a legal entity on the Canton ledger <sup>8</sup> .

The goal of the observability configuration is to make it easier to navigate through the provenance of any state or event in the wider system. Any or all of these identifiers can be used to correlate a combination of logs, metrics, state. Three of these in particular are intended to be set to corresponding business identifiers derived from your specific business domain – **application-id**, **workflow-id**, and **command-id**.

Navigation is enabled by the use of structured logs from as many components as possible<sup>9</sup>. It is recommended that your custom components likewise emit structured logs for more accurate consumption by OpenTelemetry.

<sup>6</sup> Distributed tracing is essential to efficient debugging and diagnosis of any distributed application. While technically distinct identifiers Trace and Span Ids are closely linked. If unfamiliar with their use OpenTelemetry has a good primer (<https://opentelemetry.io/docs/concepts/signals/traces/>), Grafana has a reasonable demo (<https://grafana.com/docs/tempo/latest/introduction/>), and we demonstrate their use later in this guide.

<sup>7</sup> Equivalent to “blockheight” in other public blockchains that do not support privacy. As privacy dictates that each participant node sees a different projection of the global blockchain, the offset is not comparable across different Participant Nodes. It is commonly the preferred id when dealing with a single participant node due to being a simple, monotonic, total-order on ledger events witnessed by a Participant Node.

<sup>8</sup> By virtue of their role in the ledger model, all parties are (and the associated entity must be) capable of authorizing a (sub-)transaction or ledger event. See the Daml Ledger Authorization Model for details (<https://docs.daml.com/concepts/ledger-model/ledger-integrity.html#authorization>)

<sup>9</sup> Where loggers cannot be configured to emit structured logs directly, log parsers are used to convert raw log files in the usual manner. This is primarily done in the OTEL Collector configuration.

## Direct Ledger Inspection using Correlation Identifiers

Starting from `$ make stop clean-all && make build start`, we proceed with initiating the example application app-user onboarding:

```
✓ % make create-app-install-request | cat -n
  1 docker compose -f docker/app-user-shell/compose.yaml --env-file .env run --rm
create-app-install-request || true
  2 get_token ledger-api-user AppProvider
  3 get_user_party AppProvider participant-app-provider
  4 http://participant-app-provider:7575/v2/users/AppProvider
  5 get_token ledger-api-user Org1
  6 get_user_party Org1 participant-app-user
  7 http://participant-app-user:7575/v2/users/Org1
  8 get_token administrator Org1
  9 http://validator-app-user:5003/api/validator/v0/scan-proxy/dso-party-id
10 http://participant-app-user:7575/v2/commands/submit-and-wait
11 --data-raw {
12     "commands" : [
13         { "CreateCommand" : {
14             "template_id": "#quickstart-licensing:Licensing.AppInstall:AppInstallRequest",
15             "create_arguments": {
16                 "dso": "DSO::1220015e721c8ec5c1a5868b418442f064530e367c2587a9b43bd66f58c7bfdffec4",
17                 "provider": "AppProvider::12202fe7b2bf950dca3858b880d9ee0dd58249af8821ff2330ea1b80420852e816ff",
18                 "user": "Org1::122072b20a515d939910f9412f915cff8c1a7a427ddde76c6d0b7646d0022d4d4551",
19                 "meta": { "values": []}
20             }
21         }
22     }
23
24 ],
25     "workflow_id" : "create-app-install-request",
26     "application_id": "ledger-api-user",
27     "command_id": "create-app-install-request",
28     "deduplication_period": { "Empty": {} },
29     "act_as": [ "Org1::122072b20a515d939910f9412f915cff8c1a7a427ddde76c6d0b7646d0022d4d4551" ],
30     "read_as": [ "Org1::122072b20a515d939910f9412f915cff8c1a7a427ddde76c6d0b7646d0022d4d4551" ],
31     "submission_id": "create-app-install-request",
32     "disclosed_contracts": [],
33     "domain_id": "",
34     "package_id_selection_preference": []
35   }
36 {"update_id": "1220e48d6d59af99a1b61eca414fe25766c342bb4e7d8d485e049a11a7f2267ed5c0", "completion_offset": 73}
```

This is the output of a script submitting a create command to the app-user's participant node, it already contains number of the correlation ids mentioned above:

14	Template Id	#quickstart-licensing:Licensing.AppInstall:AppInstallRequest
16-18	Party Ids	DSO:::1220015e721c8ec5c1a5868b...ddfec4 AppProvider:::12202fe7b2bf950d...e816ff Org1:::122072b20a515d939910f94...4d4551
25	Workflow Id	create-app-install-request
26	Application Id	ledger-api-user
27	Command Id	create-app-install-request
31	Submission Id	create-app-install-request
36	Transaction Id	1220e48d6d59af99a1b61eca414fe...7ed5c0

We can immediately use the transaction id in Daml Shell to view the associated ledger transaction:

```
√ % make shell
docker compose -f docker/daml-shell/compose.yaml --env-file .env run --rm daml-shell || true
Connecting to jdbc:postgresql://postgres-splice-app-provider:5432/scribe...
Connected to jdbc:postgresql://postgres-splice-app-provider:5432/scribe
postgres-splice-app-provider:5432/scribe> transaction
1220e48d6d59af99a1b61eca414fe25766c342bb4e7d8d485e049a11a7f2267ed5c0
transactionId: 1220e48d6d59af99a1b6..., offset: 48, workflowId: create-app-install-request -
Feb 17, 2025, 5:26:09 AM
+ #1220e48d6d59af99a1b6....:0 quickstart-licensing:Licensing.AppInstall:AppInstallRequest
(005c17f89b7fd1d5fde9...) {"dso":
"DSO:::1220015e721c8ec5c1a5868b418442f064530e367c2587a9b43bd66f58c7bfddfec4", "meta": {
"values": []}, "user": {
"Org1:::122072b20a515d939910f9412f915cff8c1a7a427ddde76c6d0b7646d0022d4d4551", "provider": "AppProvider:::12202fe7b2bf950dca3858b880d9ee0dd58249af8821ff2330ea1b80420852e816ff"}
postgres-splice-app-provider:5432/scribe 3f → 48>
```

From here we can get more identifiers:

Ledger Offset	48
Ledger Event Id	#122026e55e3f82e27542....:0
Contract Id	00cb53139ff0eb7ec57b...

The Workflow Id, Template Id, and Party Ids are also visible here. The ledger offset can be very useful if you are going to query PQS or the Ledger API directly for more information. The Contract Id can be used to immediately display the contract in Daml Shell:

```
postgres-splice-app-provider:5432/scribe 3f → 48> contract
005c17f89b7fd1d5fde9c548740c32924eededdacc6320256892636b4e3b7d66aaca101220777c5420863adb012c4f38847049346014c44eba7cd54bf
```

58950dd6a18679053
Identifier quickstart-licensing:Licensing.AppInstall:AppInstallRequest
Type Template
Created at 48 (not yet active)
Archived at <active>
Contract ID 005c17f89b7fd1d5fde9...
Event ID #1220e48d6d59af99a1b6...:0
Contract Key
Payload dso: DSO::1220015e721c8ec5c1a5868b418442f064530e367c2587a9b43bd66f58c7bfddfec4 meta: values: user: Org1::122072b20a515d939910f9412f915cff8c1a7a427ddde76c6d0b7646d0022d4d4551 provider: AppProvider::12202fe7b2bf950dca3858b880d9ee0dd58249af8821ff2330ea1b80420852e816ff
postgres-splice-app-provider:5432/scribe 3f → 48>

If the problem is in fact a bug in your smart contract, then exploring the transaction and related provenance within Daml Shell and utilizing the Daml IDE to synthesize and rerun the relevant transactions will normally be sufficient to identify the issue. However, if only due to the comparative lines of code, the root cause of most issues will be off ledger. Consequently, significant value in these identifiers derives from correlating these identifiers with the consolidated logs and other information collected through Open Telemetry.

## Correlated Logs and Traces using Correlation Identifiers

To advance the example, we log in as the AppProvider and accept the `AppInstallRequest`, resulting in:

Canton Network Quickstart Home AppInstallRequests AppInstalls Licenses LicenseRenewalRequests OAuth2 AppProvider Logout

## App Installs

### Existing AppInstalls

Contract ID	DSO	Provider	User	Meta	Num Licenses Created	Actions
002ac6577aa4ae... ...	DSO::1220015e72... ...	AppProvider::122... ...	Org1::122072b20... ...	{"data":{}} ...	0 ...	<button>Actions</button>

The usual browser-based developer inspection tools can extract the relevant correlation ids:

```
<!DOCTYPE html>
<!-- Notice: #####SPDX-License-Identifier: 0BSD-->
Copyright (c) 2025, Digital Asset (Switzerland) GmbH and/or its affiliates. All rights reserved.
SPDX-License-Identifier: 0BSD
-->
<html lang="en">
  <head>::</head>
  <body>
    <div id="root">
      <header>::</header>
      <main class="container mt-4">
        <div class="mt-4">
          <h2>App Installs</h2>
          <div class="mt-4">
            <h3>Existing AppInstalls</h3>
            <table class="table table-fixed">
              <thead>::</thead>
              <tbody>::</tbody>
              <tr>
                <td class="ellipsis-cell">
                  "002ac6577aa4ae996ceee4ac9c82c453128ea1ae2e9ebb383db623c40341e9cc1012204485321cc1029c1bc78934e0d133d88e398a7b09bb6c6f4b9dc191aba1c8dde9" == $0
                </td>
                <td class="ellipsis-cell">DSO::1220015e721c8ec5c1a5868b418442f064530e367c2587a9b43bd66f58c7bfddfec4</td>
                <td class="ellipsis-cell">
                  "AppProvider::12202fe7b2bf950dc3a3858b880d9ec0dd58249af8821ff2330ealb80420852e816ff"
                </td>
                <td class="ellipsis-cell">Org1::122072b20a515d939910f9412f915cff8c1a7a427ddd76c6d0b7646d0022d4d4551</td>
                <td class="ellipsis-cell">{"data":{}}</td>
                <td>0</td>
                <td>::</td>
              </tr>
            </tbody>
          </table>
        </div>
      </main>
    </div>
  </body>
</html>
```

We can also see the HTTP call to the Backend-Service when we issue a new license, and again the response to the call provides additional identifiers.

Request URL: http://localhost:3000/api/app-installs?commandId=79062314-1354-439b-b5c8-b889bec1024f

Request Method: POST

Status Code: 200 OK

Remote Address: [::]:3000

Referrer Policy: strict-origin-when-cross-origin

Response Headers:

- Cache-Control: no-cache, no-store, max-age=0, must-revalidate
- Connection: keep-alive
- Content-Type: application/json
- Date: Mon, 17 Feb 2025 05:51:02 GMT
- Expires: 0
- Pragma: no-cache
- Server: nginx/1.27.0
- Transfer-Encoding: chunked
- X-Content-Type-Options: nosniff
- X-Frame-Options: DENY
- X-Xss-Protection: 0

Request Headers:

- Accept: application/json, text/plain, \*/\*
- Accept-Encoding: gzip, deflate, br, zstd
- Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
- Connection: keep-alive
- Content-Length: 31
- Content-Type: application/json
- Cookie: JSESSIONID=A47f38953C5FC505161C3C0CBF7E935; XSRF-TOKEN=07ec0d7c-abd2-4f7c-934f-3d2b558005e2
- Host: localhost:3000
- Origin: http://localhost:3000
- Referer: http://localhost:3000/app-installs
- Sec-Ch-Ua: "Not(A 'Brand');v='99', \"Google Chrome\";v='133', \"Chromium\";v='133'
- Sec-Ch-Ua-Mobile: ?0
- Sec-Ch-Ua-Platform: "macOS"
- Sec-Fetch-Dst: self
- Sec-Fetch-Mode: cors
- Sec-Fetch-Site: same-origin
- User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
- X-XsrF-Token: 07ec0d7c-abd2-4f7c-934f-3d2b558005e2

Query String Parameters:

- commandId: 79062314-1354-439b-b5c8-b889bec1024f

Request Payload:

```
params: {meta: {data: {}}}
  params: {meta: {data: {}}}
    meta: {data: {}}
```

Response:

```
{
  "installId": "002ac6577aa4aee9906cee4aec9c82c453128ea1ae2e9ebb383db623c40341ee9cca1012204485321cc1029c1bc78934e0d133d88e398a7b09bb6c6f4b9dc191aba1c8dde0",
  "licenseId": "00333b5474b4de92ccb9f1ae51d95f94eca249c8b4dc5852b0b419d28051d81d3dca1012208708d250c2a48862bb48b6873f73f99e9072900fc0f8e8f89eaf6988c224bd"
}
```

Command Id **79062314-1354-439b-b5c8-b889bec1024f**

Contract Id **AppInstall** **002ac6577aa4aee9906cee4aec9c82c45312...**

Contract Id **License** **79062314-1354-439b-b5c8-b889bec1024f**

As we have already seen, contract ids can be used in Daml Shell to inspect the contracts directly. In addition, due to the way the OpenAPI interface for the Backend has been designed, the Command Id is visible as a query parameter to the POST. We can use this to query the consolidated logs in Grafana:

```

2025-02-17 15:55:48.000 [nginx-app-provider] GET /api/app-installs/002ac6577aa4aee9906cee4aec9c82c453128ea1ae2e9ebb383db623c40341ee9cca1012204485321cc1029c1bc78934e0d133d88e398
    a7099bb6c6f4b9dc191aba1c8dde9/create-license?commandId=79062314-1354-439b-b5c8-b889bec1024f HTTP/1.1 405
2025-02-17 15:51:02.794 [participant] Responding with updates., context: {commandId -> '79062314-1354-439b-b5c8-b889bec1024f', updateId -> '122053c509d405e77eb680a859bd017a06
    cb5068581d6e84794f489816c42d10bb', workflowId -> '', offset -> '136'}
2025-02-17 15:51:02.798 [participant] Responding with update trees., context: {commandId -> '79062314-1354-439b-b5c8-b889bec1024f', updateId -> '122053c509d405e77eb680a859bd0
    17a66ccb5068581d6e84794f489816c42d10bb', workflowId -> '', offset -> '136'}
2025-02-17 15:51:02.771 [participant] Updated command deduplication data for Hash(f886d6a7c59a527797529fedce22a813386f03ac669853c867a1bed4d82f0f) to CommandDeduplicationData(
    changeId = ChangedId(applicationId = AppProvider, commandId = 79062314-1354-439b-b5c8-b889bec1024f, act as = AppProvider::12202fe7b2f...),
    latestDefiniteAnswer = DefiniteAnswerEvent(
        offset = 136,
        publicationTime = 2025-02-17T05:51:02.734637Z,
        submissionId = Some(0bb83701c-855a-45f1-885d-ddef8bd7a5a3),
        traceContext = TraceContext(traceId = 442fd29567f04e2fa3f8d1dc9cf51628-421077c4bf2eaa-01)
    ),
    latestAcceptance = DefiniteAnswerEvent(
        offset = 136,
        publicationTime = 2025-02-17T05:51:02.734637Z,
        submissionId = Some(0bb83701c-855a-45f1-885d-ddef8bd7a5a3),
        traceContext = TraceContext(traceId = 442fd29567f04e2fa3f8d1dc9cf51628-421077c4bf2eaa-01)
    )
)
2025-02-17 15:51:02.731 [participant] Phase 7: Storing at offset=00000000000000000000 TransactionAccepted(
    recordedTime = 2025-02-17T05:51:02.116504Z
)

```

The command-id has provided logs from the App-Provider's Nginx reverse proxy in front of the backend and their Participant Node. We can verify the Nginx log matches the request we saw from the browser:

Fields	Value
body_bytes_sent	319
container_id	f1701453e6757e1a1905ece12fc56aaa5ac0862c5712fe0bf77d58573a2a9d9a
container_name	/nginx-app-provider
fluent_tag	f1701453e675
http_referrer	http://localhost:3000/app-installs
http_user_agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.3
remote_addr	6
remote_user	172.20.0.1
request	POST /api/app-installs/002ac6577aa4aee9906cee4aec9c82c453128ea1ae2e9ebb383db623c40341ee9cca1012204485321cc1029c1bc78934e0d133d88e398     a7099bb6c6f4b9dc191aba1c8dde9/create-license?commandId=79062314-1354-439b-b5c8-b889bec1024f HTTP/1.1 200
request_time	0.950
service_name	nginx-app-provider
source	stdout
status	200
time_local	17/Feb/2025:05:51:02 +0000

Critically, we can also see in the same aggregated log the entries that indicate the Participant Node submitting the transaction to the Canton Synchronization Domain:

```
v 2025-02-17 15:51:02.096 [participant] Phase 1 completed: Submitting 3 envelopes for Transaction request, submitters AppProvider::12202fe7b2bf..., command-id 7906  
2314-1354-439b-b5c8-b889bec1024f
```

Fields	
__timestamp	2025-02-17T05:51:02.096Z
__version	1
container_id	f5e24ff03c2853f0c77160621e09b653eff08f131efcd38d444141f53284d7e1
container_name	/participant-app-provider
detected_level	info
fluent_tag	f5e24ff03c28
level	INFO
level_value	20000
logger_name	c.d.c.p.p.TransactionProcessor:participant=participant/domainId=global-domain::1220015e721c
message	Phase 1 completed: Submitting 3 envelopes for Transaction request, submitters AppProvider::12202fe7b2bf..., command-id 79062314-1354-439b-b5c8-b889bec1024f
service_name	participant
source	stdout
thread_name	canton-env-ec-33
trace_id	442fd29567f04e2fa3f8d1dc9cf51628
Links	
TraceID	442fd29567f04e2fa3f8d1dc9cf51628

Tempo

Was notified that the transaction was successfully committed to the Canton Ledger:

```

v 2025-02-17 15:51:02.723 [participant] Finalizing Transaction request=2025-02-17T05:51:02.116504Z with event Some(
    TransactionAccepted(
        recordTime = 2025-02-17T05:51:02.116504Z,
        updateId = 122053c509d405e7eab680a8559bd017a06cb5068581d6e84794f409816c42d10bb,
        transactionMeta = TransactionMeta(ledgerEffectiveTime = 2025-02-17T05:51:01.971445Z, submissionTime = 2025-02-17T05:51:01.971445Z,
        ...),
        completion = CompletionInfo(
            actAs = AppProvider::12202fe7b2bf...
            commandId = 79062314-1354-439b-b5c8-b889bec1024f,
            applicationId = AppProvider,
            deduplication period = (offset=Offset(Bytes(0000000000000000001))),
            submissionId = Some(0b837b1c-855a-45f1-885d-ddef0bd7a5a3),
            ...
        ),
        nodes = 3,
        roots = 1,
        ...
    )
).

Fields
  _timestamp          2025-02-17T05:51:02.723Z
  _version             1
  container_id        f5e24ff03c2853f0c77160621e09b653eff08f131efcd38d444141f53284d7e1
  container_name       /participant-app-provider
  detected_level      info
  fluent_tag          f5e24ff03c28
  level                INFO
  level_value          20000
  logger_name          c.d.c.p.p.TransactionProcessor:participant=participant/domainId=global-domain::1220015e721c
  Finalizing Transaction request=2025-02-17T05:51:02.116504Z with event Some(
    TransactionAccepted(
        recordTime = 2025-02-17T05:51:02.116504Z,
        updateId = 122053c509d405e7eab680a8559bd017a06cb5068581d6e84794f409816c42d10bb,
        transactionMeta = TransactionMeta(ledgerEffectiveTime = 2025-02-17T05:51:01.971445Z, submissionTime = 2025-02-17
        T05:51:01.971445Z, ...),
        completion = CompletionInfo(
            actAs = AppProvider::12202fe7b2bf...
            commandId = 79062314-1354-439b-b5c8-b889bec1024f,
            applicationId = AppProvider,
            deduplication period = (offset=Offset(Bytes(0000000000000000001))),
            submissionId = Some(0b837b1c-855a-45f1-885d-ddef0bd7a5a3),
            ...
        ),
        nodes = 3,
        roots = 1,
        ...
    )
).
  message
  service_name         participant
  source               stdout
  thread_name          canton-env-ec-36
  trace_id             442fd29567f04e2fa3f8d1dc9cf51628
Links
  TraceID              442fd29567f04e2fa3f8d1dc9cf51628

```

Tempo

And finally added to the App-Provider's local ledger:<sup>10</sup>

---

<sup>10</sup> This is an example of an important feature of the Canton Network. The participant node is only aware of the existence of this transaction because it is authorized to be informed of the transaction by the relevant Daml Smart Contracts and the privacy semantics of the Daml Ledger Model. Privacy is guaranteed, not because the contract data is obscured as ciphertext; but, because the ledger model ensures participants without a verified right to know do not receive the transaction in any form.

```

v 2025-02-17 15:51:02.731 [participant] Phase 7: Storing at offset=000000000000000088 TransactionAccepted(
    recordTime = 2025-02-17T05:51:02.116504Z,
    updateId = 122053c509d405e77eab680a8559bd017a06cb5068581d6e84794f409816c42d10bb,
    transactionMeta = TransactionMeta(ledgerEffectiveTime = 2025-02-17T05:51:01.971445Z, submissionTime = 2025-02-17T05:51:01.971445Z,
    ...),
    completion = CompletionInfo(
        actAs = AppProvider::12202fe7b2bf...
        commandId = 79062314-1354-439b-b5c8-b889bec1024f,
        applicationId = AppProvider,
        deduplication period = (offset=Offset(Bytes(00000000000000000001))),
        submissionId = Some(0b837b1c-855a-45f1-885d-ddef0bd7a5a3),
        ...
    ),
    nodes = 3,
    roots = 1,
    ...
)

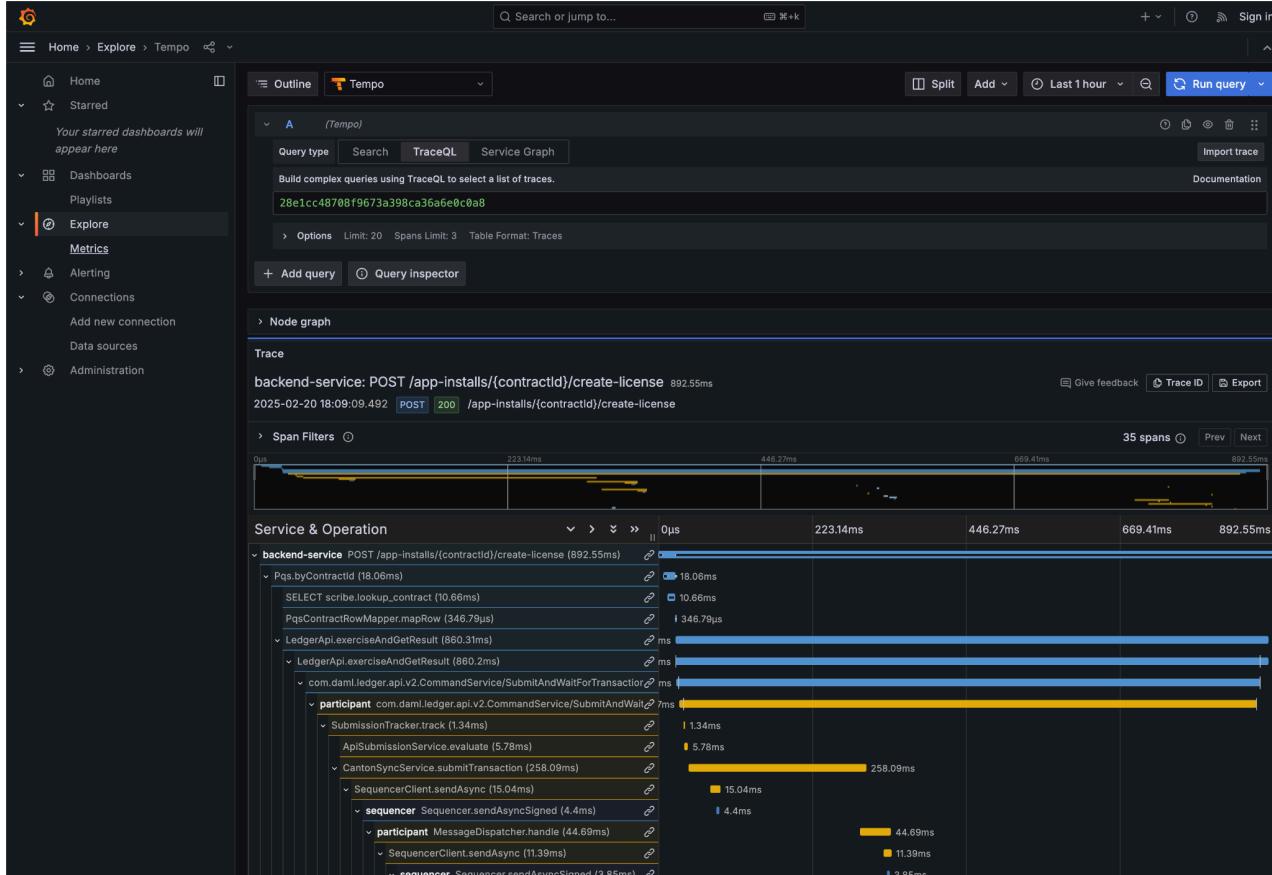
Fields
  _timestamp          2025-02-17T05:51:02.731Z
  _version            1
  container_id       f5e24ff03c2853f0c77160621e09b653eff08f131efcd38d444141f53284d7e1
  container_name     /participant-app-provider
  detected_level    info
  fluent_tag         f5e24ff03c28
  level              INFO
  level_value        20000
  logger_name        c.d.c.p.i.p.ParallelIndexerSubscription:participant=participant
Phase 7: Storing at offset=000000000000000088 TransactionAccepted(
    recordTime = 2025-02-17T05:51:02.116504Z,
    updateId = 122053c509d405e77eab680a8559bd017a06cb5068581d6e84794f409816c42d10bb,
    transactionMeta = TransactionMeta(ledgerEffectiveTime = 2025-02-17T05:51:01.971445Z, submissionTime = 2025-02-17T05:51:01.971445Z, ...),
    completion = CompletionInfo(
        actAs = AppProvider::12202fe7b2bf...
        commandId = 79062314-1354-439b-b5c8-b889bec1024f,
        applicationId = AppProvider,
        deduplication period = (offset=Offset(Bytes(00000000000000000001))),
        submissionId = Some(0b837b1c-855a-45f1-885d-ddef0bd7a5a3),
        ...
    ),
    nodes = 3,
    roots = 1,
    ...
)
  message
    participant
  service_name        participant
  source              stdout
  thread_name         input-mapping-pool-7
  trace_id            442fd29567f04e2fa3f8d1dc9cf51628
Links
  TraceID            442fd29567f04e2fa3f8d1dc9cf51628

```

Note that from these we can obtain additional correlation ids, any of which could have been used to find these log lines:

Ledger Offset	000000000000000088
Transaction Id	122053c509d405e77eab680a855...2d10bb
Submission Id	0b837b1c-855a-45f1-885d-ddef0bd7a5a3
Trace Id	442fd29567f04e2fa3f8d1dc9cf51628

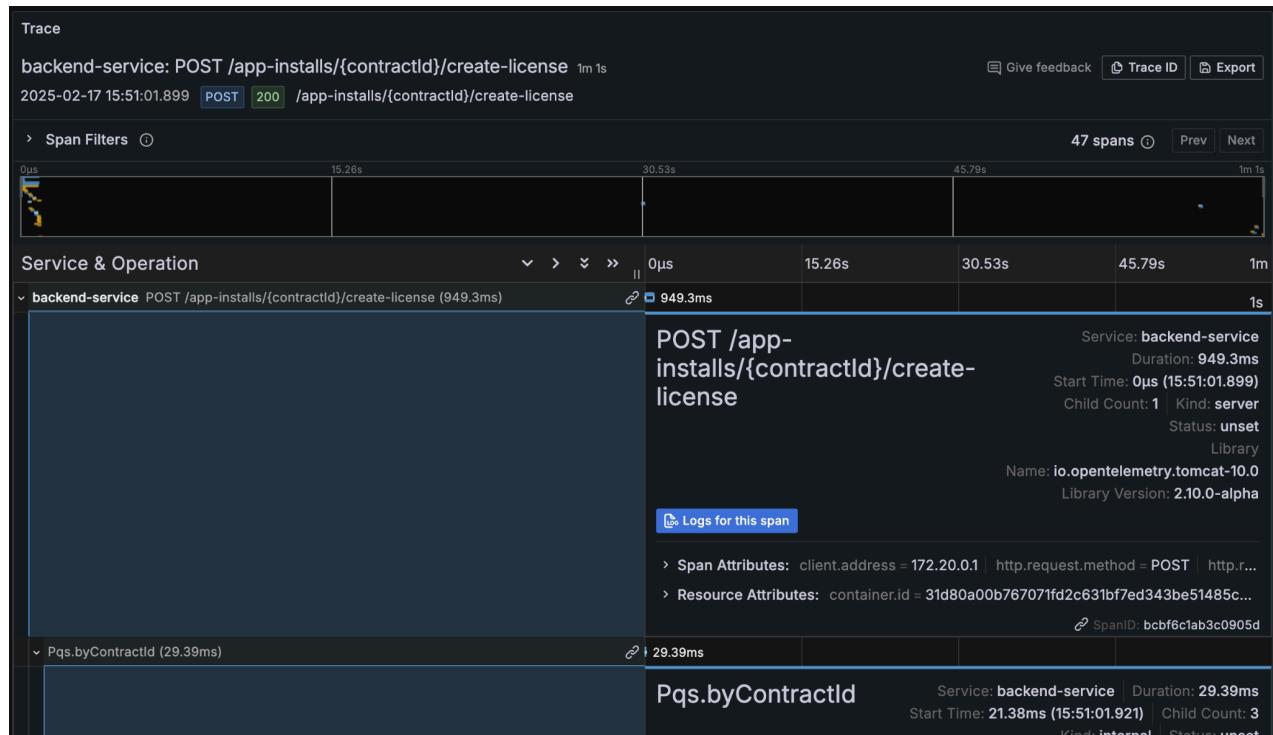
In particular the Trace Id is invaluable because it can link us directly into Tempo to see the distributed operation spans:



Here we can see the flow of the create license operation behind the backend reverse proxy:

- Initial POST handler in the Backend Service
- Backend query against PQS to retrieve the AppInstall contract
- Call to the App-Provider Ledger API from the Backend Service
- Preparation of the Transaction by the Participant Node and submission to the Canton Network

One very powerful aspect of the Grafana suite is the degree to which it integrates the various observability tools in the quickstart stack. We have already seen this with the link from the consolidated logs to Tempo; however, it also runs the other way. Expanding a span in Tempo provides a link to “Logs for this span”.



These link to the logs for the specific component (backend-service, participant, sequencer, etc) correlated to this span.

Using different correlation ids can allow us to navigate and explore the history of our distributed application. We have seen the transaction committed to the ACS within the participant node; however, PQS also logs identifiers associated with the transactions it indexes.

The transactionId and the traceId can both be used to broaden our understanding of the create-license backend operation and what followed after.

Services All Levels All Search 28e1cc48708f9673a39f Trace ID Enter variable value

Logs

```

> 2025-02-28 18:09:10.377 [pq] Converting transaction 1220dbe96001996fb1489d002338b96f1c1efaf9af20656acfff3514c5f8a552 (offset: 78, events: 3, remote trace: 28e1cc48708f9673a398ca36a6e0c0a8)

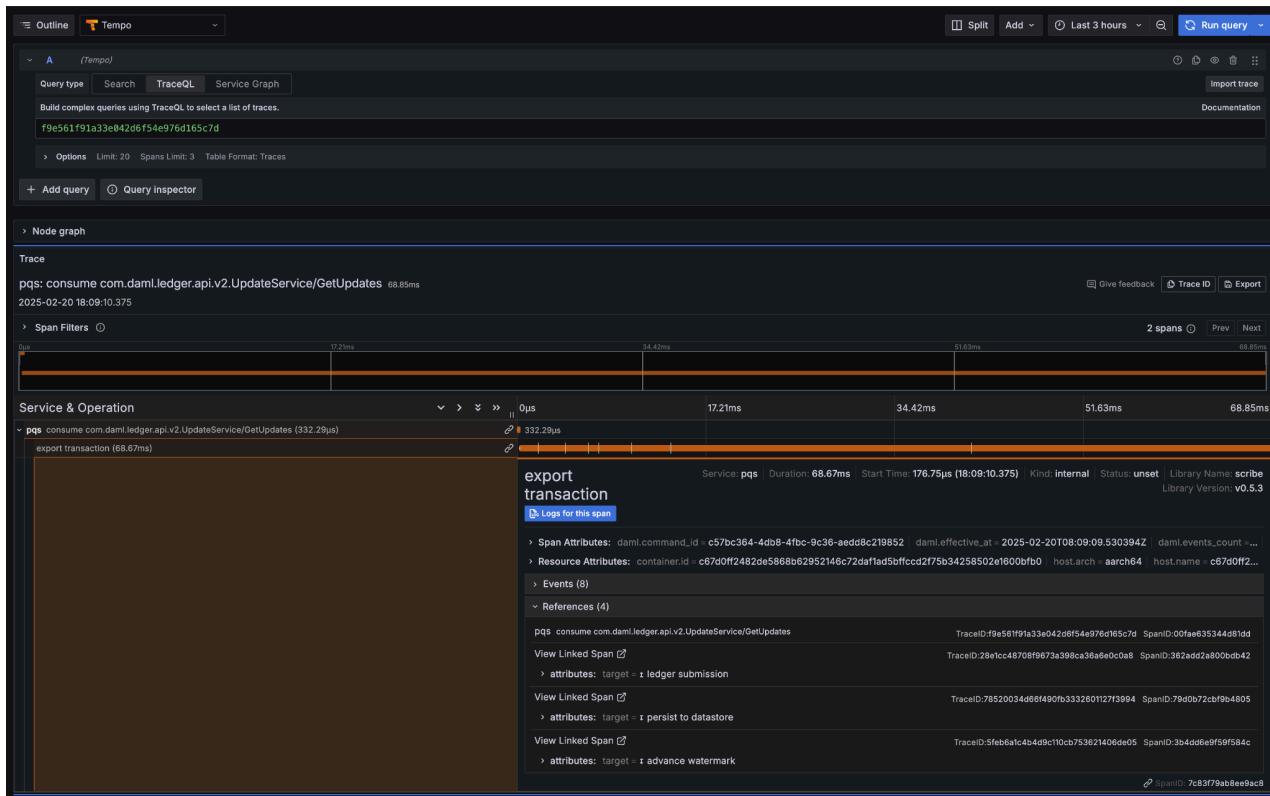
Fields
  application      scribe
  container_id    c67d0ff2482de5868b62952146c72daf1ad5bfffccdf75b34258592e1600bf0
  correlation_id  28e1cc48708f9673a398ca36a6e0c0a8
  fiberId         [246873724]
  flags           1
  host_arch       aarch64
  host_name        c67d0ff2482d
  level            info
  location         com.digitalasset.zio.daml.ledgerapi.UpdateService.process:176
  observed_timestamp 1740038950377274292
  os_description   Linux 6.10.14-linuxkit
  os_type          linux
  process_command_args  [/usr/lib/jvm/java-17-openjdk-arm64/bin/java", "-jar", "scribe.jar", "pipeline", "ledger", "postgres-document"]
  process_executable_path /usr/lib/jvm/java-17-openjdk-arm64/bin/java
  process_pid       1
  process_runtime_description Debian OpenJDK 64-Bit Server VM 17.0.12+7-Debian-2deb11u1
  process_runtime_name OpenJDK Runtime Environment
  process_runtime_version 17.0.12+7-Debian-2deb11u1
  scope_name        com.digitalasset.zio.daml.ledgerapi.UpdateService
  service_instance_id 8aa8479b-7036-4459-9745-bd01fd0abf0b
  service_name      pq
  service_version   v0.5.3
  severity_number   9
  span_id           7c83f79ab8ee9ac8
  telemetry_distro_name opentelemetry-java-instrumentation
  telemetry_distro_version 2.6.0
  telemetry_sdk_language java
  telemetry_sdk_name opentelemetry
  telemetry_sdk_version 1.49.0
  threadName        zio-fiber-246873724
  trace_id          f9e561f91a33e042d6f54e976d165c7d

Links
  TraceID          f9e561f91a33e042d6f54e976d165c7d Tempo

> 2025-02-28 18:09:10.360 [participant] Request com.daml.ledger.api.v2.CommandService/SubmitAndWaitForTransactionTree by /172.20.0.37:60408: sending response
  Request tid:28e1cc48708f9673a398ca36a6e0c0a8
> 2025-02-28 18:09:10.351 [participant] Updated command deduplication data for Hash(519a91d00b206c908ea15e3de7f4cf4dcc4ba9a3e5f44bfe302737919984fa3) to CommandDeduplicationData(
  change id = ChangeId(application Id = AppProvider, command Id = c57bc364-4db8-4fbc-9c36-aedd8c219852, act as = AppProvider::1220a28bace9...),
  latest definite answer = DefiniteAnswerEvent(
    offset = 78,
    publication time = 2025-02-28T08:09:10.333111Z,
    submission id = Some(bb12e860-884d-4298-8b3c-4a54fd2fedd),
  )
)

```

PQS ingestion is a distinct operation performed by a background process. The traceId for this log is therefore distinct; however it still links back to the trace and transaction identifiers associated with the ledger data it is ingesting. You can see this if you follow the Tempo link:



The expanded “references” section in the “export transaction” span include links to traces for related PQS processes and also, critically, the trace for command submission that resulted in the transaction. The link takes us directly to that trace, which in this case is the same one we just came from.

Querying and navigating through correlated logs, traces, and spans makes understanding the multiple moving parts involved in a Canton Network Application much easier. Keep in mind that you can only navigate logs and traces that have been emitted; and, query identifiers that have been included or attached. Therefore we highly recommend you periodically take the time to look for opportunities to enrich and expand the logging within your application.

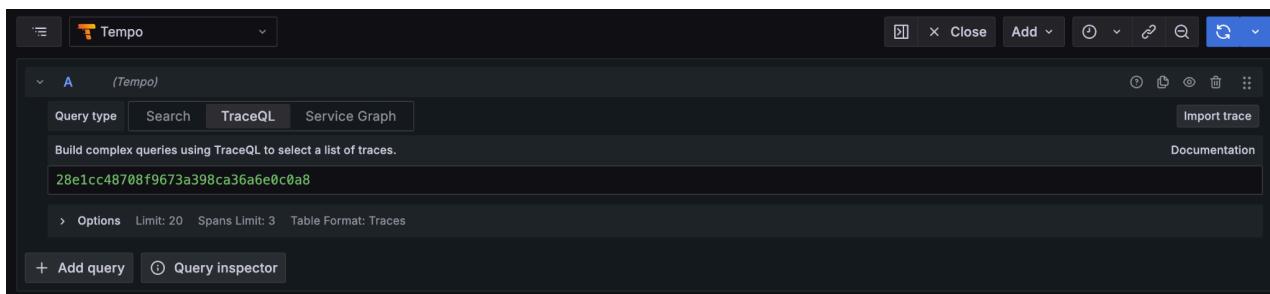
One final thing that isn't visible immediately, but is whenever you hover over any log line is the option to view the log context for that line:

The screenshot shows a Grafana dashboard titled "Quickstart - consolidated logs". The left sidebar has a "Logs" section expanded, showing various service categories like Home, Starred, Dashboards, Explore, Alerting, Connections, Administration, and Metrics. The main panel displays a log search results table with columns for Fields, Value, and Count. One log entry is highlighted in red, showing a POST request to "/api/app-installs" from an Nginx app provider. The log details include fields such as body\_bytes\_sent (319), container\_id (f1701459e6757e1a1985ece12fc56aaa5ac0862c5712fe0bf77d58573a2a9d9a), container\_name (/nginx-app-provider), fluent\_tag (f1701453e675), http\_referrer (http://localhost:3000/app-installs), http\_user\_agent (Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_15\_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36), remote\_addr (172.20.0.1), remote\_user (POST /api/app-installs/002ac6577aa4aee9906cee4aec9c82c453128ea1ae2e9ebb383db623c48341ee9cca1012204485321cc1029c1bc78934e0d13d88e398a7b09bb6c6f4b9dc191abac18dde9), and request (POST /api/app-installs/002ac6577aa4aee9906cee4aec9c82c453128ea1ae2e9ebb383db623c48341ee9cca1012204485321cc1029c1bc78934e0d13d88e398a7b09bb6c6f4b9dc191abac18dde9). A "Show context" button is visible next to the highlighted log entry.

This will pop up a window with a full unfiltered view of the component's logs for that time, with the relevant line highlighted. In the case of the Nginix log line, this provides a single click view of the other traffic being served at the same time:

The screenshot shows a "Log context" modal. At the top, it says "Widen the search" and "Refine the search". Below that is a "Choose" dropdown and an "Include LogQL pipeline operations" toggle. The main area displays several log entries for "nginx-app-provider" with timestamps ranging from 2025-02-17 15:51:08.000 to 2025-02-17 15:51:02.000. The logs show various requests to "/api/app-installs" with different status codes (200, 208) and body byte counts (319, 0). The modal has "Wrap lines" and "Center matched line" buttons at the bottom.

It is also worth keeping in mind that Grafana exposes access to the raw queries for Tempo and Loki, and also Prometheus (not shown). It is well worth the time to experiment with these and discover how to probe the unified metrics, traces, and logs available via the observability stack:



A starting point for finding documentation on these see:

- Loki: <https://grafana.com/docs/loki/latest/query/>
- Tempo: <https://grafana.com/docs/tempo/latest/traceql/>
- Prometheus:  
<https://grafana.com/docs/grafana/latest/datasources/prometheus/query-editor/>