TypeDB 3.0 is live! Get started for free.

Meet TypeDB and TypeQL

TypeDB is a polymorphic database with a conceptual data model, a strong subtyping system, a symbolic reasoning engine, and a beautiful and elegant type-theoretic language: TypeQL.

Visit learning center

Read philosophy

Conceptual Modeling

Entity-Relation-Attribute

TypeQL implements the Enhanced Entity-Relation-Attribute model for its schemas and data. Entities, relations, and attributes are all first-class citizens and subtypable, allowing for expressive modeling without normalization or reification.

```

define

  id sub attribute, value string;
  email sub id;
  path sub id;
  name sub id;

  user sub entity,
    owns email @unique,
    plays permission:subject,
    plays request:requester;
```

Declarative Schema

The schema provides a structural blueprint for data organization, ensuring referential integrity in production. Extend your data model seamlessly in TypeDB, maintaining integrity during model updates and avoiding any query rewrites or code refactors.

```

define

  full-name sub attribute, value string;
  id sub attribute, value string;
  email sub id;
  employee-id sub id;

  user sub entity,
    owns full-name,
    owns email @unique;
  employee sub user,
```

Abstract Types

Define abstract entity, relation, and attribute types in your schema to extend concrete types from. Build templates with ownership of abstract attributes and playing of abstract roles for subtypes to extend and override.

```

define

  id sub attribute, abstract, value string;
  email sub id;
  path sub id;

  user sub entity, abstract,
    owns id;
  employee sub user,
    owns email as id;
  resource sub entity, abstract,
```

Type Inheritance

Type inheritance in TypeDB allows you to create new types based on existing ones, providing hierarchy and abstraction in your data model. By inheriting attributes and relationships from parent types, schema design is simplified, promoting reusability and consistency.

```

define
  user sub entity,
    owns full-name,
    owns email;
  intern sub user;
  employee sub user,
    owns employee-id,
    owns title;
  part-time-employee sub employee,
    owns weekly-hours;
```

Strong Type System

Type Inference

TypeDB's type inference resolves queries against the schema to generate polymorphic results. Queries on supertypes automatically return results for subtypes, and the types of variables can even be omitted to match only the shape of the data.

```

define
  user sub entity, abstract,
    owns id,
    plays resource-ownership:owner;
  employee sub user, owns employee-id as id;
  resource sub entity, abstract,
    owns id,
    plays resource-ownership:resource;
  file sub resource, owns path as id;
  database sub resource, owns name as id;
  commit sub resource, owns hash as id;
```

Semantic Validation

TypeDB validates all queries and rules against the type system defined in the schema to ensure semantic correctness. Nonsensical writes are automatically blocked, and nonsensical reads throw an exception instead of returning an empty result set.

```

define
  weekly-hours sub attribute value long;
  full-time-employee sub employee;
  part-time-employee sub employee, owns weekly-hours;

  insert
    $francois isa full-time-employee,
      has full-name "François Durand",
      has email "francois@typedb.com",
      has employee-id 184,
      has weekly-hours 35;
```

Symbolic Reasoning

Rule-Based Reasoning

TypeDB's symbolic reasoning enables the automated deduction of new facts and relationships based on existing data and rules you define. Rule chaining and branching allow complex behavior to arise from simple rules, creating rich, high-level insights.

```

define

  rule transitive-team-membership:
    when {
      (team: $team-1, member: $team-2) isa team-membership;
      (team: $team-2, member: $member) isa team-membership;
    } then {
      (team: $team-1, member: $member) isa team-membership;
    };

  rule inherited-team-permission:
```

Explanations

TypeDB's reasoning engine functions on deductive reasoning, so inferred data can always be traced back to its source. Perform root-cause analysis using TypeDB's Explanations feature, guaranteeing accountability of generated data.

```

query = "match $perm isa inherited-permission; get;"

with open_session.transaction(TransactionType.READ) as tx:
    results = tx.query().get(query)
    for result in results:
        inherited_permission = result.explainables().relation("perm")
        explanations = tx.query().explain(inherited_permission)
        for explanation in explanations:
            condition = explanation.condition()
            rule = explanation.rule()
```

Polymorphic Queries

Variablized Types

Schema types and relation roles can be variablized in addition to data instances, making schema querying as easy as data querying. Queries can contain both schema and data constraints, allowing for patterns that represent highly complex conceptual structures.

```

define
  user sub entity, has full-name,
    plays mentorship:mentor,
    plays mentorship:trainee;
  employee sub user;
  contractor sub user;
  mentorship sub relation,
    relates mentor,
    relates trainee;

match
```

Inheritance Polymorphism

TypeQL implements inheritance polymorphism, allowing subtypes to inherit the behaviors of the supertypes they extend, whether concrete or abstract. Write TypeQL queries that return results with a common supertype, without enumerating the subtypes.

```

define
  user sub entity,
    owns full-name,
    owns email @unique;
  employee sub user,
    owns employee-id @key;

insert
  $john isa employee,
    has full-name "John Doe",
    has email "john@typedb.com",
```

Interface Polymorphism

Ensure conceptual consistency between defined types and their behaviors in perfect parallel to your object model by harnessing TypeQL's interface polymorphism. Types can own the same attributes and play the same roles, even if they share no common supertypes.

```

define
  name sub attribute, value string;
  user sub entity, owns name;
  team sub entity, owns name;
  table sub entity, owns name;

match
  $x has name $n;
fetch
  $n;

```

Parametric Polymorphism

Write queries that create or delete data instances without specifying their types by utilizing parametric polymorphism. Queries are resolved against the schema when run, allowing them to write data of multiple types matching declared properties.

```

match
```

```
  $data isa $T;
  $data has data-expiration-date < 2023-09-27;
delete
  $data isa $T;
```

Modern Language

Near Natural

Due to its OOP properties and simple syntax, queries written in TypeQL read close to natural language. Domain experts and non-technical users alike can quickly grasp the intent of a query, reducing the learning curve and making query maintenance a breeze.

```

match
  $kevin isa user, has email "kevin@typedb.com";
insert
  $chloe isa part-time-employee,
    has full-name "Chloé Dupond",
    has email "chloe@typedb.com",
    has employee-id 185,
    has weekly-hours 35;
  $hire (employee: $chloe, ceo: $kevin) isa hiring,
    has date 2023-09-27;

```

Fully Declarative

TypeQL is fully declarative, allowing you to define query patterns without considering execution strategy. TypeDB's query planner always deconstructs queries into the most optimized plans, so you never have to think about the logical implementation.

```

define
  user sub entity,
    owns full-name,
    owns email;
  intern sub user;
  employee sub user,
    owns employee-id;
  full-time-employee sub employee;
  part-time-employee sub employee,
    owns weekly-hours;
```

Composable Patterns

Patterns in TypeQL are fully composable. Every complex pattern can be broken down into a conjunction of atomic constraints, which can be concatenated in any order. Any pattern composed of valid constraints is guaranteed to be valid itself, no matter how complex.

```

match
  $user isa user;
fetch
  $user: full-name;

match
  $user isa user;
  $user has email "john@typedb.com";
fetch
  $user: full-name;

```

Nested Subqueries

Search for complex data structures with a single query and network trip using nested subqueries. Retrieve results for nested queries as a list or perform aggregations over them, including results for optional attribute matches.

```

match
  $user isa user;
fetch
  $user: email, full-name, employee-id;
  teams: {
    match
      (team: $team, member: $user) isa team-membership;
    fetch
      $team: name;
  };
```

Structured Results

Query results can be serialized for easy consumption in your application with TypeQL's native JSON outputs. Switch from an asynchronous answer stream to a single structured collection, and define the result format using projections in the query structure.

```

match
  $user isa full-time-employee;
fetch
  $user as employee: attribute;
limit 1;

# JSON output:
[{
    "employee": {
        "type": { "root": "entity", "label": "full-time-employee" },
        "attribute": [
```

Aggregates and Expressions

Perform basic mathematical operations directly in your queries or rules with aggregations and arithmetic expressions, enabling dynamic and efficient data computation.

```

match
  $user isa user;
  $perm (subject: $user) isa permission;
group $user;
get;
count;

match
  $dir isa directory,
    has path $path,
    has size $kb;
```

Query Builder

Use the TypeQL query builder to auto-generate queries using a code-first approach in Java or Rust, with other languages coming soon. This permits the generation of TypeDB queries through a robust and streamlined process.

```

TypeQLMatch.Filtered builtQuery = TypeQL.match(
    cVar("user").isa("user").has("full-name", "Kevin Morrison"),
    cVar("file").isa("file").has("path", cVar("path")),
    cVar("perm").rel(cVar("user")).rel(cVar("file")).isa("permission")
).get(cVar("path"));

// builtQuery =
// match
//   $user isa user, has full-name 'Kevin Morrison';
//   $file isa file, has path $path;
//   $perm ($user, $file) isa permission;
```

Query Templates

3.0 Roadmap

Build query templates that accept a tuple of attribute values as parameters and execute them repeatedly for lists of supplied values. The template is stored in the transaction cache, reducing network load and ensuring sanitization of input strings.

[illegible]

Expressive Relations

N-ary Relations

Construct rich data representations by directly implementing unary, binary, ternary, and n-ary relations in your conceptual model. TypeQL's expressivity allows you to use the same constructor format for all relations, regardless of the number of roleplayers.

```

match
  $omar isa contractor, has email "omar@typedb.com";
insert
  $term (user: $omar) isa user-termination,
    has termination-date 2023-09-19,
    has termination-reason "end of contract";

match
  $naomi isa user, has email "naomi@typedb.com";
  $eng isa group, has name "Engineering";
insert
```

Nested Relations

Relations are first-class citizens in TypeQL and so can own attributes and play roles in other relations just like entities. With no limit to the depth of nesting for relations, you can express the full richness of your data without reifying your data model.

```

match
  $john isa user, has email "john@typedb.com";
  $readme isa file, has path "/usr/johndoe/repos/typedb/readme.md";
  $edit isa action, has name "edit file";
  $perm (subject: $john, object: $readme, action: $edit) isa permission;
  $kevin isa user, has email "kevin@typedb.com";
insert
  $rqst (target: $perm, requestee: $kevin) isa change-request,
    has requested-change "revoke";


```

Variadic Relations

With TypeQL's expressive relation constructor, you can easily implement relations where the same roleplayer plays multiple roles, multiple roleplayers play the same role, or a combination of both. Read queries always return all matched roleplayers.

```

match
  $submit isa action, has name "submit order";
  $approve isa action, has name "approve order";
insert
  (segregated-action: $submit, segregated-action: $approve) isa segregation-policy;

match
  $kevin isa user, has email "kevin@typedb.com";
insert
  (reviewer: $kevin, reviewee: $kevin) isa permission-review;

```

Cardinality Constraints

3.0 Roadmap

All attributes and relations have many-to-many cardinality by default. Apply constraints in the schema to apply stricter cardinalities wherever needed, with the expressivity to select a single value or a specific range.

```

define

  name sub attribute, value string;
  object-type sub attribute, value string;

  action sub entity,
    owns name @card(1),
    owns object-type @card(1,*)
    plays segregation-policy:segregated-action @card(0,*);

  segregation-policy sub relation,
```

Intuitive Attributes

Multi-Valued Attributes

TypeQL is a conceptual data modeling language, and all attributes have many-to-many cardinality by default. Giving an entity or relation multiple attributes of the same type is as simple as declaring them in an insert, and read queries automatically return all values.

```

insert
  $john isa full-time-employee,
    has primary-email "john.doe@typedb.com",
    has email "j.doe@typedb.com",
    has email "john@typedb.com",
    has email "sales@typedb.com";




```

Globally Unique Attributes

Attributes are globally unique in TypeQL. If two entities each have an attribute with the same type and value, then they both have the same attribute instance. This allows for highly efficient data traversals, keeps disk usage low, and maintains a consistent model.

```

insert
  $roadmap isa file,
    has path "/typedb/feature-roadmap.pdf",
    has confidentiality "public";
  $cloud isa repository,
    has name "typedb-cloud",
    has confidentiality "restricted";
  $sales isa database,
    has name "sales",
    has confidentiality "restricted";

```

No Nulls

Unlike SQL and NoSQL modeling languages, TypeQL is entirely conceptual and does not need to implement nulls to store the absence of a value. Keep nulls out of your query results without compromising for a schema-less database.

```

insert
```

```
  $john isa user, has full-name "John Doe";
  $david isa user, has email "david@typedb.com";

match
  $user isa user;
fetch
  $user: full-name, email;

# JSON output:
[{
```

Attribute Constraints

Define a key constraint on an attribute to make ownership of that attribute required and ensure a unique value. Alternatively, use a unique constraint instead to ensure uniqueness without requiring ownership. Apply regex constraints to string attributes to enforce defined patterns.

```

define
  full-name sub attribute, value string;
  office-location sub attribute,
    value string,
    regex "^(London|Paris|Dublin)$";
  id sub attribute, value string;
  email sub id, regex "^(.+)@(\\S+)$";
  employee-id sub id;
  user sub entity,
    owns full-name,
    owns email @unique;
```

Purely Abstract Attributes

3.0 Roadmap

Define abstract attribute types with no declared value type, and extend them to define subtypes with different value types. Easily retrieve attribute values of different types by querying the abstract supertype.

```

define
  id sub attribute, abstract;
  email sub id, value string;
  employee-id sub id, value long;
  path sub id, value string;
  user sub entity, owns email;
  employee sub user, owns employee-id;
  resource sub entity, abstract, owns id;
  file sub resource, owns path as id;

match
```

Compound Value Types

3.0 Roadmap

Define compound value types for your attributes constructed from primitive types.

```
[illegible]
```

Native Language Drivers

Rust

Docs › GitHub › Crates ›

```
cargo add typedb-driver
```

Python

Docs › GitHub › PyPI ›

```
pip install typedb-driver
```

Node.js

Docs › GitHub › NPM ›

```
npm install typedb-driver
```

Java

Docs › GitHub › Repo ›

```
com.vaticle.typedb:typedb-driver
```

C#

Docs › GitHub › NuGet ›

```
dotnet add package TypeDB.Driver
```

C++

Docs › GitHub ›

```
#include <typedb.hpp>
```

C

Docs › GitHub ›

```
#include <typedb_driver.h>
```

Go

Coming Soon

GitHub ›

```
[illegible]
```

Request your language

Submit request on GitHub >

Robust API

ACID Compliance

TypeDB is ideal for complex and highly transactional applications due to its ACID compliance, ensuring data integrity and reliability. This prevents corruption and guarantees consistency during concurrent transactions.

Asynchronous Protocol

Communication with TypeDB servers is fully asynchronous, preventing any bottlenecks due to query processing. Send queries concurrently from the client, taking only as long as the network round-trip, while the server continues to process the received queries concurrently.

Reactive Streaming

TypeDB servers return query results in a stream as they are discovered. This enables the application to consume the results lazily for built-in vertical parallelism. Answer streaming is reactive, allowing the server to optimize for the rate of consumption and network latency.

Stateful & Programmatic

TypeDB's network API streamlines integration with the database with fine control over connections, sessions, and transactions. Maximize performance based on your use case with automatic cache optimization, from high-speed bulk loading to high-volume reads.

Rust	Python	Node.js	Java	C	C++	C#

```

let connection = Connection::new_core("localhost:1729")?;
let databases = DatabaseManager::new(connection);

databases.create("access-management-db").await?;

let session = Session::new(databases.get("access-management-db").await?, SessionType::
let tx = session.transaction(TransactionType::Write).await?;
tx.query().define(access_management_schema).await?;
tx.commit().await?;
drop(session);

```

Programmatic Schema Migration

Transactional Mutations

Execute complex schema mutations with ACID guarantees by using TypeDB's stateful API. Mutations are executed via schema-write transactions, ensuring large-scale migrations can be carried out in production without inconsistent states or downtime.

Existing Data Validation

All schema mutation operations can be performed with data in place. By utilizing transactions, existing data can enter illegal states within the scope of the transaction, as long as they are resolved by commit. Any remaining inconsistencies will trigger a rollback.

Type Hierarchy Modification

The type hierarchies declared in the schema are entirely mutable. New types can be defined, existing types can be undefined, and the supertype of any type can be changed to any other type, with inherited behaviors automatically re-assigned where necessary.

Type Behavior Modification

Modify the behaviors of types with complete flexibility. Rename types, assign or de-assign ownership of attributes, create or remove roles from relations, and add or remove the ability for types to play roles, with new behaviors automatically inherited in hierarchies.

Rust | Python | Node.js | Java | C | C++ | C#

```rust

let connection = Connection::new_core("localhost:1729")?;
let databases = DatabaseManager::new(connection);
let session = Session::new(databases.get("access-management-db").await?, SessionType::
let tx = session.transaction(TransactionType::Write).await?;

// create a new abstract type "user"
let mut user = tx.concept().put_entity_type("user".to_owned()).await?;
user.set_abstract(&tx).await?;

// change the supertype of "employee" to "user"
let mut employee = tx.concept().get_entity_type("employee".to_owned()).await?.unwrap()
```

Beautiful IDE

Dedicated Environment

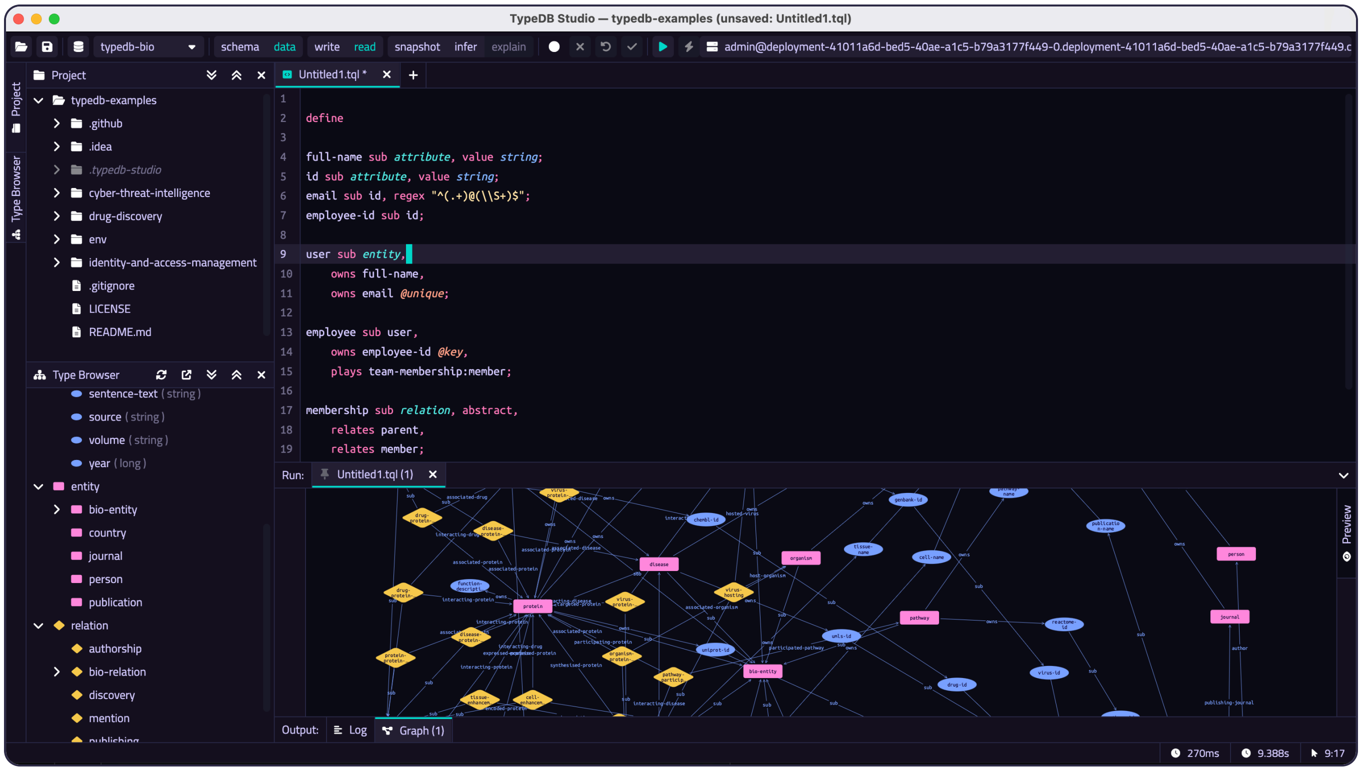
TypeDB Studio is a cross-platform IDE for developing with TypeDB on Windows, macOS, and Linux. Manage databases, build and execute queries, and explore results with an integrated experience.

Schema Manager

Use TypeDB Studio's schema manager to simplify schema editing, with a graphical interface for defining and visualizing the types in your data model and their properties. Create, maintain, explore, and extend schemas with ease.

Data Visualizer

TypeDB Studio features an interactive visualizer for query results in a hypergraph format. Explore your schemas and data, and discover the root causes of data inferred by the reasoning engine in an easy-to-access visual manner using TypeDB's Explanations.



```
define

full-name sub attribute, value string;
id sub attribute, value string;
email sub id, regex "^(.+)@(\\S+)$";
employee-id sub id;

user sub entity,
    owns full-name,
    owns email @unique;

employee sub user,
    owns employee-id @key,
    plays team-membership:member;

membership sub relation, abstract,
    relates parent,
    relates member;
```

Powerful CLI

Database Administration

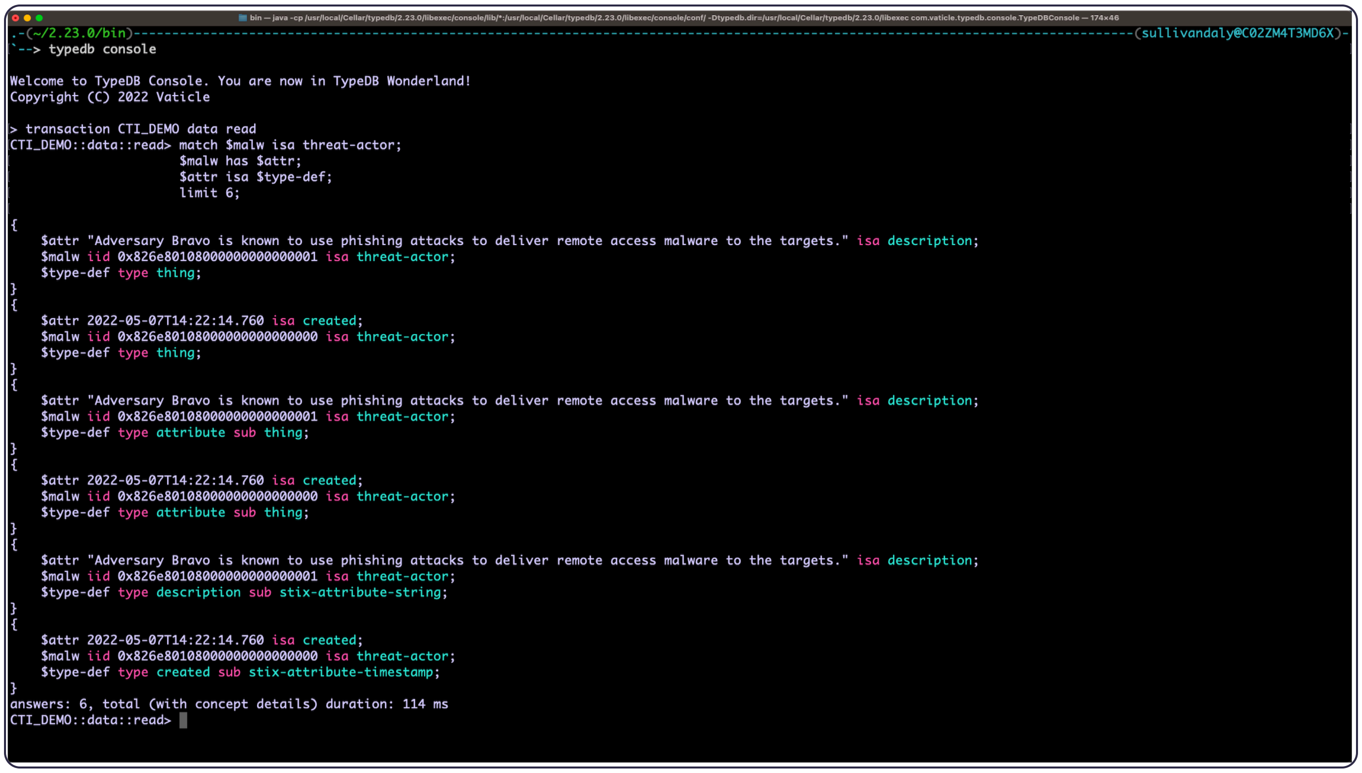
Access essential database management tools directly from the command line with TypeDB Console. Create and delete databases, build and extend schemas, or define and undefine rules with ease.

Access Control

TypeDB Console simplifies user access control by providing a powerful command line tool, allowing administrators to easily grant or restrict access, and reset their passwords.

Data Management

TypeDB Console allows you to easily insert, modify, and query your data, enabling you to interface directly with the database to rapidly fix issues in production.



bin — java -cp /usr/local/Cellar/typedb/2.23.0/libexec/console/lib/*:/usr/local/Cellar/typedb/2.23.0/libexec/console/conf/ -Dtypedb.dir=/usr/local/Cellar/typedb/2.23.0/libexec com.vaticle.typedb.console.TypeDBConsole — 174×46

```
.-(~/2.23.0/bin)-------------------------------------------------------------------------------------------------------------------------(sullivandaly@C02ZM4T3MD6X)-
`--> typedb console

Welcome to TypeDB Console. You are now in TypeDB Wonderland!
Copyright (C) 2022 Vaticle

> transaction CTI_DEMO data read
CTI_DEMO::data::read> match $malw isa threat-actor;
                      $malw has $attr;
                      $attr isa $type-def;
                      limit 6;

{
    $attr "Adversary Bravo is known to use phishing attacks to deliver remote access malware to the targets." isa description;
    $malw iid 0x826e80108000000000000001 isa threat-actor;
    $type-def type thing;
}
{
    $attr 2022-05-07T14:22:14.760 isa created;
    $malw iid 0x826e80108000000000000000 isa threat-actor;
    $type-def type thing;
}
{
    $attr "Adversary Bravo is known to use phishing attacks to deliver remote access malware to the targets." isa description;
    $malw iid 0x826e80108000000000000001 isa threat-actor;
    $type-def type attribute sub thing;
}
{
    $attr 2022-05-07T14:22:14.760 isa created;
    $malw iid 0x826e80108000000000000000 isa threat-actor;
    $type-def type attribute sub thing;
}
{
    $attr "Adversary Bravo is known to use phishing attacks to deliver remote access malware to the targets." isa description;
    $malw iid 0x826e80108000000000000001 isa threat-actor;
    $type-def type description sub stix-attribute-string;
}
{
    $attr 2022-05-07T14:22:14.760 isa created;
    $malw iid 0x826e80108000000000000000 isa threat-actor;
    $type-def type created sub stix-attribute-timestamp;
}
answers: 6, total (with concept details) duration: 114 ms
CTI_DEMO::data::read> 
```

Resilient Clustering

Automated Failover

TypeDB runs in clusters consisting of multiple server nodes. They offer automatic failover, ensuring continual service by seamlessly switching to a backup node if the primary becomes unresponsive. This drastically minimizes downtime and prevents data loss.

Automated Load Balancing

Active-active clustering in TypeDB distributes read transactions across all active nodes to evenly balance the workload. This ensures efficient use of resources, strong consistency, and high availability.

Resilient Sessions

All TypeDB clients have built-in session resiliency for communication with TypeDB clusters. If a client has an open session to a node that goes down, the session is seamlessly reallocated to another node when a new transaction is opened.

Cluster Resizing

3.0 Roadmap

Easily resize TypeDB clusters by seamlessly adding or removing nodes without any downtime, using the CLI for TypeDB Enterprise and GUI for TypeDB Cloud. TypeDB Cloud clusters can also be configured to resize automatically based on load.

Secured Environment

User Authentication

TypeDB includes robust username & password authentication, with customizable policies for controlling password length, complexity, and expiration.

Network Encryption

TypeDB ensures ironclad security by employing TLS 1.3 with user-provided certificates, safeguarding client-server communication with robust network encryption.

Audit Logs

The security of TypeDB is enhanced through audit logs, keeping track of authentication events, user and database management events, sessions and transactions opened, and queries executed.

User Authorization

3.0 Roadmap

Restrict database management and querying permissions granted to users in your organization by assigning them roles.

Easy Management

Fully Managed

Deploy TypeDB in two minutes with the click of a button in our fully managed cloud platform. With scalable clusters, rest easy in the knowledge that nodes will automatically recover in the event of failure.

Cloud Agnostic

TypeDB offers global coverage on AWS, GCP and Azure, ensuring seamless accessibility and optimal performance for your data-driven success. And for clusters outside the free quota, connect your cloud provider's Marketplace account for seamless billing.

Collaborative Tools

Manage data across your users, teams, and organizations with TypeDB's collaborative tools. Create projects to group clusters and control project access on a per-user or per-team basis.

Full Web Interface

TypeDB simplifies cluster management with a real-time web interface, a powerful control panel for deployment and control of managed clusters. Manage clusters, projects, teams, and users. Access support in your browser.

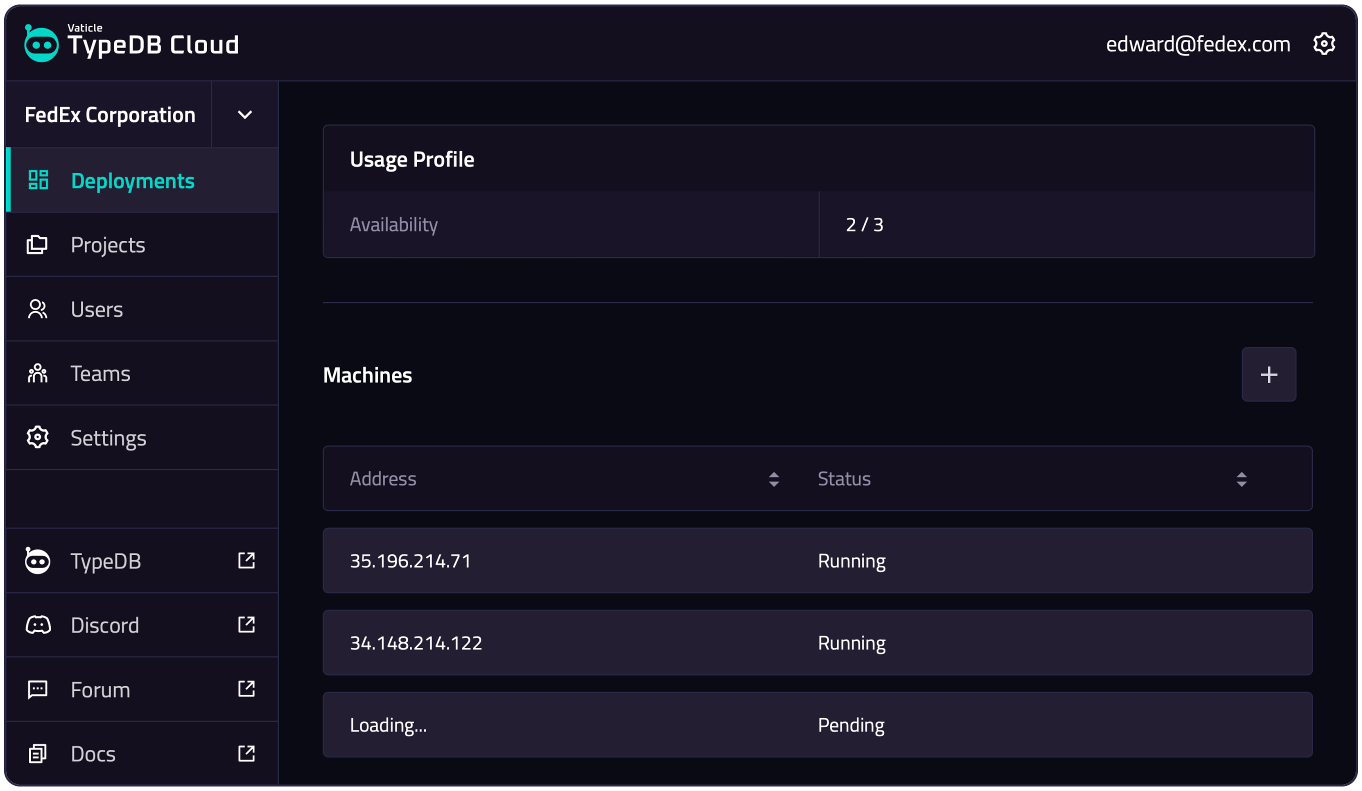
Seamless Upgrades

Upgrade your TypeDB databases to the most recent version with zero downtime and with all schema and data migrations automatically handled.

Seamless Backups

3.0 Roadmap

Protect your critical data by backing up and restoring TypeDB databases using the TypeDB Cloud CLI or web interface.



Usage Profile

Availability	2 / 3

Address	Status
35.196.214.71	Running
34.148.214.122	Running
Loading...	Pending

Get started today

Deploy in the cloud and start building now, or explore more about TypeDB and how its unique capabilities as a polymorphic database can refine and empower your applications.

Start building

A polymorphic database with a conceptual data model, a strong subtyping system, a symbolic reasoning engine, and a type-theoretic language is minutes away.

Deploy

Learn More

Read philosophy >

Visit learning center >

Subscribe to newsletter >

Join Discord community >

Subscribe to Newsletter

Connect

Discuss on Forum

Chat on Discord

Contact Us

Provide Feedback

Technology	Documentation	Resources	Company
Philosophy	Overview	Lectures	LinkedIn
Features	Manual	Papers	Careers
Cloud	TypeDB Drivers	Blog	Privacy Policy
	TypeQL		Terms of Service