Datomic Cloud Documentation

Search

- Home ›
- What is Datomic? ›
- Architecture
- Support
- Forum

Datomic Architecture

Datomic's data model - based on immutable facts stored over time - enables a physical design that is fundamentally different from traditional RDBMSs. Instead of processing all requests in a single server component, Datomic distributes transactions, queries, indexing, and caching to provide high availability, horizontal scaling, and elasticity. Datomic also allows for dynamic assignment of compute resources to tasks **without** any kind of pre-assignment or sharding.

Day of Datomic Cloud discusses Datomic application Architecture as well as Datomic's Architecture.

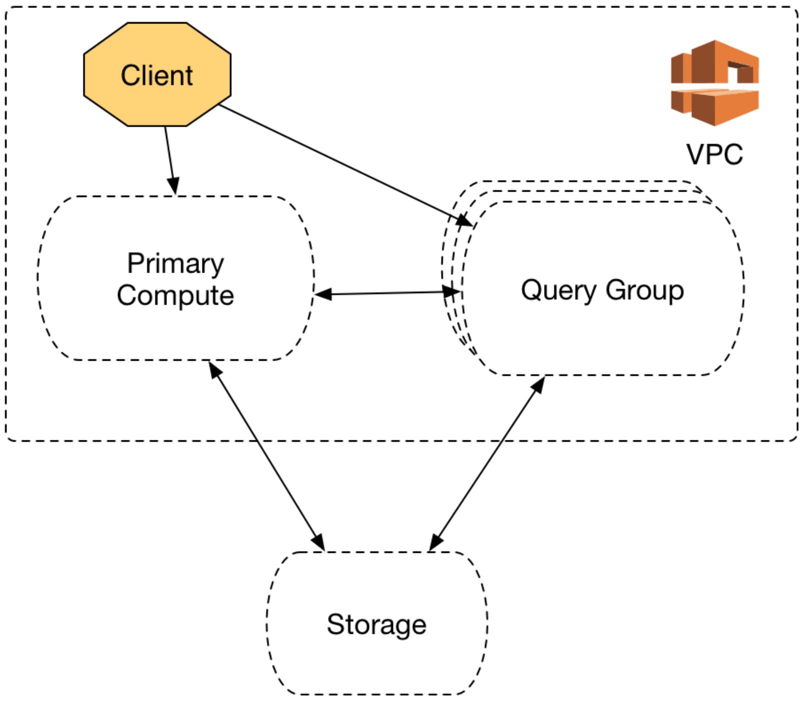
System

A complete Datomic installation is called a system. A system consists of:

- one set of Storage Resources

plus one or more compute groups:

- one Primary Compute Stack
- optional Query Groups

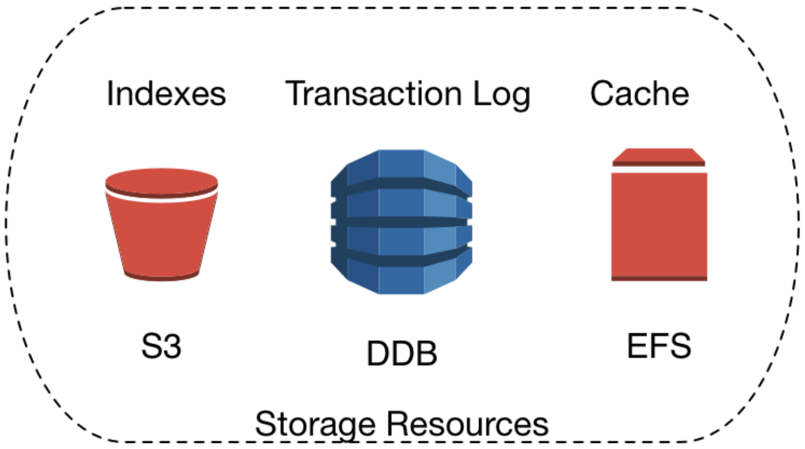


Storage Resources

The durable elements managed by Datomic are called Storage Resources, including:

- the DynamoDB Transaction Log
- S3 storage of Indexes
- an EFS cache layer
- operational logs
- A VPC and subnets in which computational resources will run

These resources are retained even when no computational resources are active, so you can shut down all the active elements of Datomic while maintaining your data.



How Datomic Uses Storage

Datomic leverages the attributes of multiple AWS storage options to satisfy its semantic and performance characteristics. As indicated in the tables below, different AWS storage services provide different latencies, costs, and semantic behaviors.

Datomic utilizes a stratified approach to provide high performance, low cost, and strong reliability guarantees. Specifically:

- ACID semantics are ensured via conditional writes with DynamoDB
- S3 provides highly reliable low cost persistence
- EFS and EC2 instance SSD storage provide very fast local caching

Stratified Durability

Purpose	Technology
ACID	DynamoDB
Storage of Record	S3
Cache	Memory > SSD > EFS
Reliability	S3 + DDB + EFS

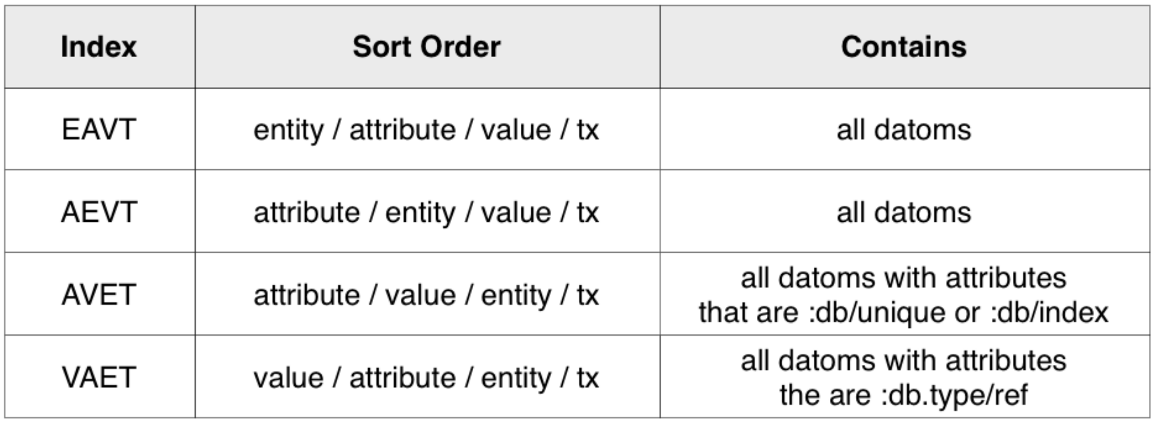
Technology	Properties
DynamoDB	low-latency CAS
S3	low-cost, high reliability
EFS	durable cache survives restarts
Memory & SSD	speed

This multi-layered persistence architecture ensures high reliability, as data missing from any given layer can be recovered from deeper within the stack, as well as excellent cache locality and latency via the multi-level distributed cache.

Indexes

Datomic indexes are covering indexes. This means the index actually contains the datoms, rather than just a pointer to them. So, when you find datoms in the index, you get the datoms themselves, not just a reference to where they live. This allows Datomic to very efficiently access datoms through its indexes.

Datomic maintains four indexes that contain ordered sets of datoms. Each of these indexes is named based on the sort order used. E, A, and V are always sorted in ascending order, while T is always in descending order:



Index	Sort Order	Contains
EAVT	entity / attribute / value / tx	all datoms
AEVT	attribute / entity / value / tx	all datoms
AVET	attribute / value / entity / tx	all datoms with attributes that are :db/unique or :db/index
VAET	value / attribute / entity / tx	all datoms with attributes the are :db.type/ref

Indexes Accumulate Only

Datomic is accumulate-only. Information accumulates over time, and change is represented by accumulating the new, not by modifying or removing the old. For example, "removing" occurs not by taking something away, but by adding a new retraction.

At the implementation level, this means that index and log segments are immutable, and can be consumed directly without coordination by any processes in a Datomic system. All Datomic processes are *peers* in that they have equivalent access to the information in the system.

Note that accumulate-only is a semantic property, and is **not** the same as append-only, which is a structural property describing how data is written. Datomic is not an append-only system, and does not have the performance characteristics associated with append-only systems.

Index Efficiency

The Datomic API presents indexes to consumers as sorted sets of datoms or of transactions. However, Datomic is designed for efficient writing at transaction time, and for use with data sets much larger than can fit in memory. To meet these objectives, Datomic:

- Stores indexes as shallow trees of segments, where each segment typically contains thousands of datoms.
- Stores segments, not raw datoms, in storage.
- Updates the datom trees only occasionally, via background indexing jobs.
- Uses an adaptive indexing algorithm that has a sub-linear relationship with total database size.
- Merges index trees with an in-memory representation of recent change so that all processes see up-to-date indexes.

- Updates the log for every transaction (the D in ACID)
- Optimizes log writes using additional data structures tuned to allow O(1) storage writes per transaction.

Primary Compute Stack

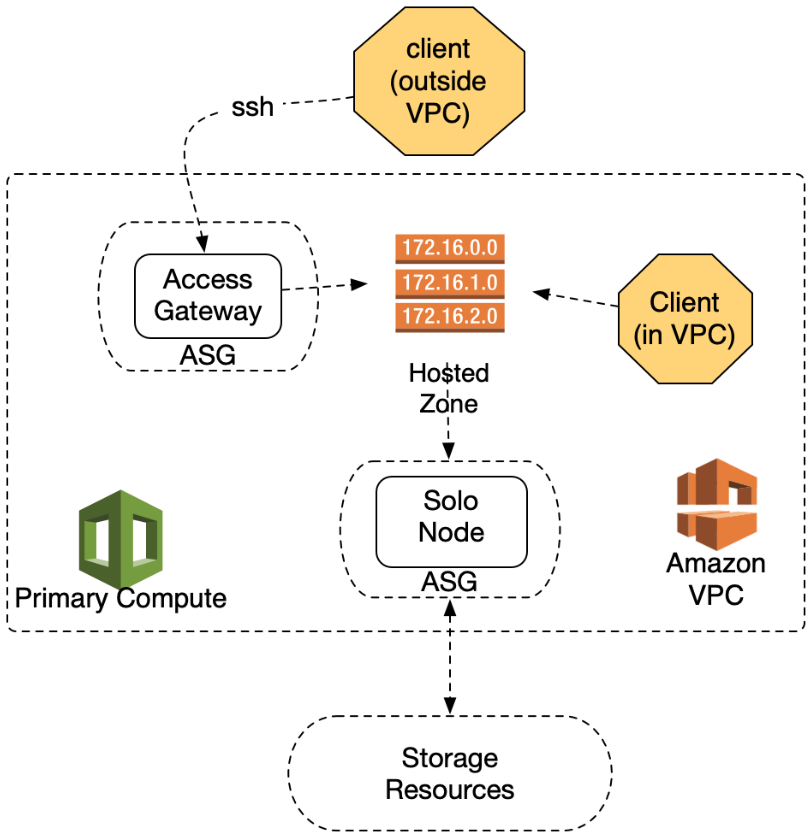
Every running system has a single primary compute stack which provides computational resources and a means to access those resources. A Primary Compute Stack consists of:

- a primary compute group dedicated to transactions, indexing, and caching.
- Route53 and/or Network Load Balancer (NLB) endpoints
- An Access Gateway Server

Topologies

The specific composition of the primary compute group is determined by your choice of Topology: either Solo or Production. The Datomic programming model is entirely the same in both Solo and Production Topologies.

Solo Topology

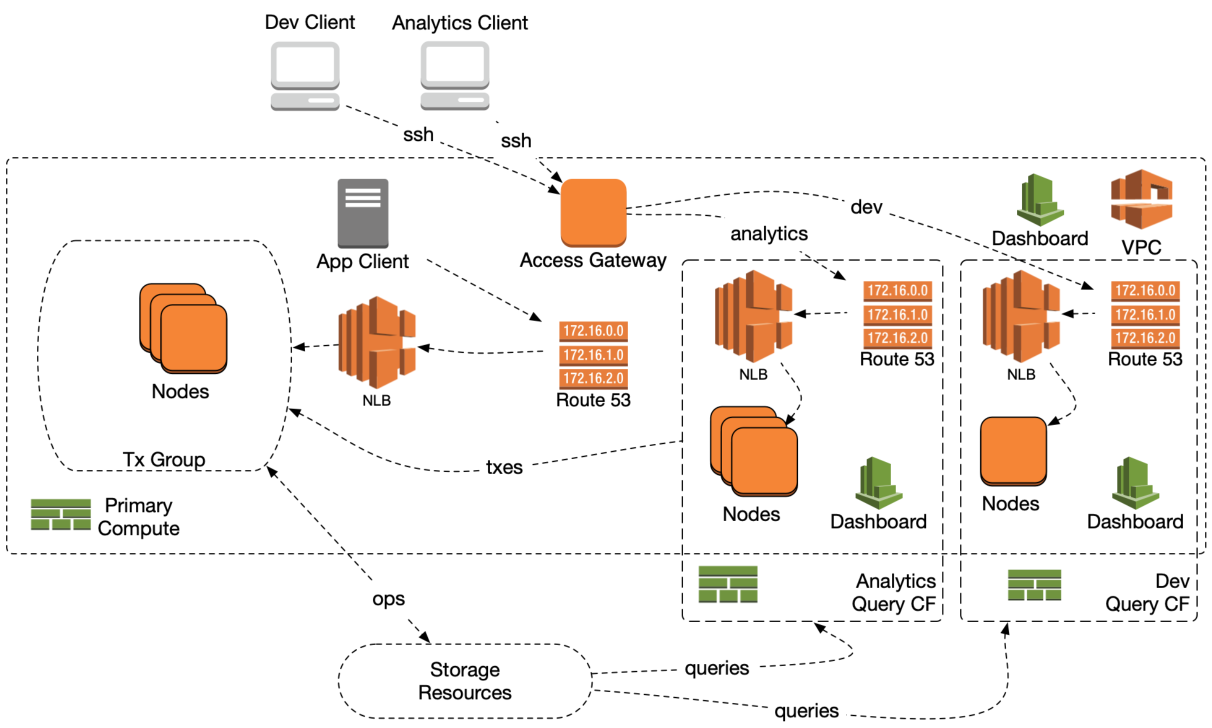


The Solo Topology provides an inexpensive way to access Datomic's full programming model for development, testing, and personal projects. The Solo Topology includes Storage Resources plus a Primary Compute Stack with:

- a dedicated VPC
- a Route53 endpoint

- a single t3.small Node
- An Access Gateway Server

Production Topology



The Production Topology includes Storage Resources plus a Primary Compute Stack with:

- a Route53 endpoint
- an Application Load Balancer for high availability (HA)
- Two or more i3.large Nodes
- An Access Gateway Server

The Production Topology also allows for the addition of optional query groups.

Query Groups

Data outlives code, and database systems often serve more than one application. Each application can have its own:

- Code
- Computational requirements
- Cache-able working set

A query group is an independent unit of computation and caching that is a distinct application deployment target. Each query group:

- Extends the abilities of an existing production topology system
- Is a deployment target for its own distinct application code
- Has its own clustered nodes

- Manages its own working set cache
- Can *elastically* auto-scale application reads without any up-front planning or sharding

Query groups deliver the entire semantic model of Datomic. In particular:

- Client code does not know or care whether it is talking to the primary compute group or to a query group.
- Query groups read Datomic data at memory speeds, just as the primary compute group does.

You can add, modify, or remove query groups at any time. For example, you might initially release a transactional application that uses only a primary compute group. Later, you might decide to split out multiple query groups:

- an autoscaling query group for transactional load
- a fixed query group with one large instance for analytic queries
- a fixed query group with a smaller instance for support

Nodes

Nodes are the computational resources for a Datomic System. Nodes provide

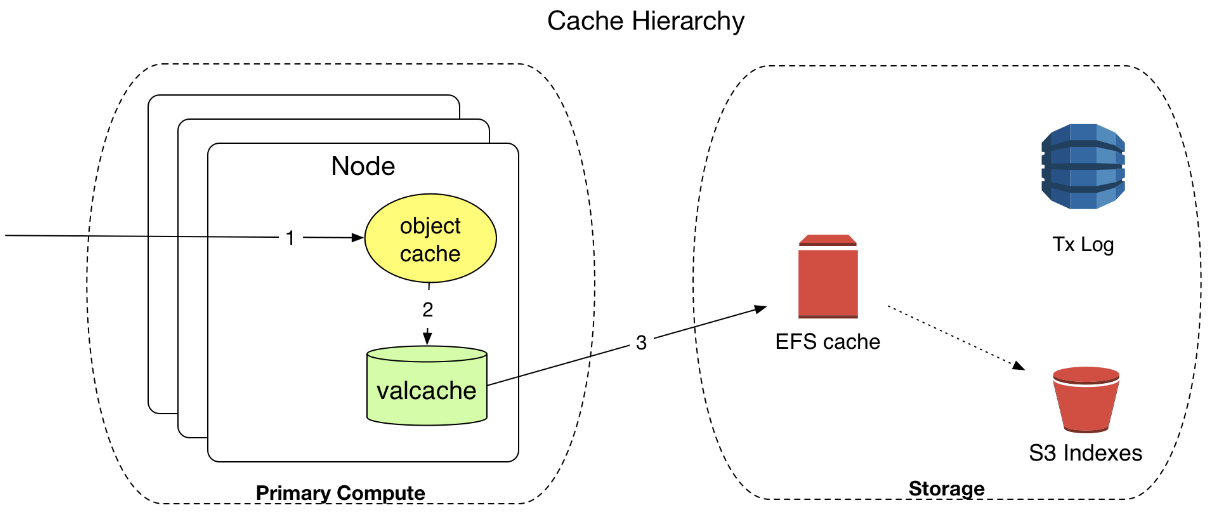
- ACID transactions
- Background indexing
- Low-latency caching
- Datalog query
- Ions

Caching

Datomic caches improve performance without requiring any action by developers or operators. Datomic's caches:

- require no configuration
- are transparent to application code
- contain segments of index or log, typically a few thousand datoms per segment
- contain only immutable data
- are always consistent
- are for performance only, and have no bearing on transactional guarantees

Datomic's cache hierarchy includes the object cache, valcache, and EFS cache.



Object Cache

Nodes maintain an LRU cache of segments as Java objects. When a node needs a segment, Datomic looks in the object cache first. If a segment is unavailable in the object cache, Datomic will look next to the valcache.

Because each process maintains its own object cache, a process will automatically adjust over time to its workload.

Valcache

Primary Compute Nodes maintain a *valcache*. Valcache implements an immutable subset of the memcached API and maintains an LRU cache backed by fast local SSDs.

If a segment is unavailable in valcache, Datomic will look next to the EFS cache.

EFS Cache

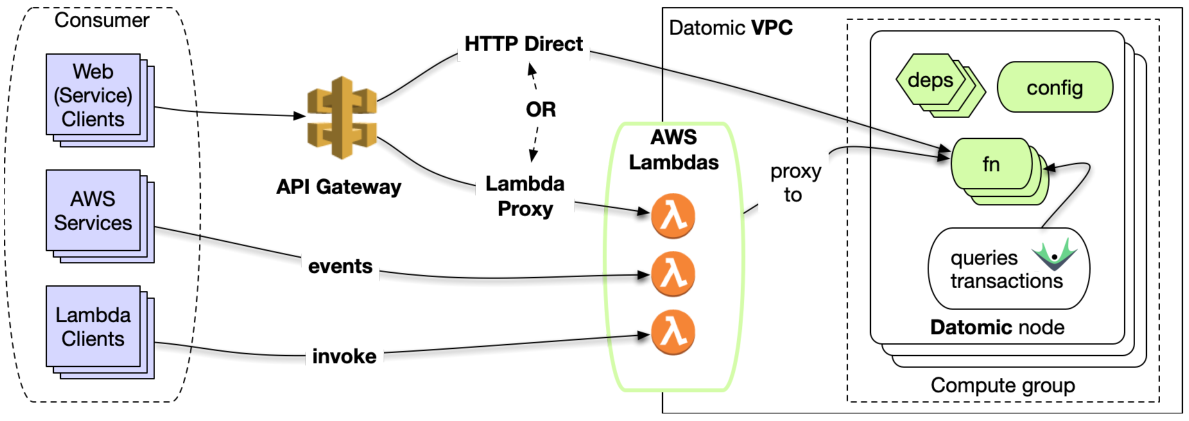
The EFS cache contains the entirety of all indexes for all databases. Datomic uses the EFS cache to populate the smaller and faster valcache and object cache, without the latency of reading from S3.

If a segment is unavailable in the EFS cache, Datomic will load the segment from S3 and repair the EFS cache.

Applications

Datomic ions provide a complete solution for Clojure application deployment on AWS. In particular, you can:

- Develop and test with real-time feedback at a local REPL.
- Rapidly deploy to AWS with no downtime.
- Reproducibly deploy across different development stages.
- Deploy multiple applications that share a common Datomic system.
- Elastically scale your entire application instead of many separate elements.
- Automatically generate AWS Lambda entry points without writing any Lambda code.
- Implement web services directly in Datomic behind AWS API Gateway.



Security

Datomic is designed to follow AWS security best practices, including:

- All authorization is performed using AWS HMAC, with key transfer via S3, enabling access control governed by IAM roles.
- All data in Datomic is encrypted at rest using AWS KMS.
- All Datomic resources are isolated in a private VPC, with optional access through a network access gateway.
- EC2 instances run in an IAM role configured for least privilege.

Access Gateway

For security, Datomic nodes all run inside a dedicated VPC that is not accessible from the internet. The *access gateway* runs inside the VPC, and provides a configurable point of access for users outside the VPC.

- You create the access gateway via an option to the CloudFormation template that creates Datomic's primary compute stack
- You then open the gateway to an SSH keypair/IP range combination.

When you run analytics support, the access gateway also acts as an analytics server.

Transit

Remote Client API implementations use wire protocol built on Transit, an open-source data interchange format.

See datom.An atomic unit of work in a database. All Datomic writes are transactional, fully serialized, and ACID (Atomic, Consistent, Isolated, and Durable).A complete Datomic installation, consisting of storage resources, a primary compute stack, and optional query groups.The durable elements managed by a Datomic system.Something that can be said about an entity. An attribute has a name, e.g. =:firstName=, and a value type, e.g. =:db.type/long=, and a cardinality.Sorted collection of datoms. Indexes are named by the order in which datom components are used for sort, e.g. An index that sorts first by entity, then attribute, then value, then tx is called EAVT.Indexes store datoms as a tree of segments, where the leaf nodes contain a few thousand datoms each.Sorted collection of datoms. Indexes are named by the order in which datom components are used for sort, e.g. An index that sorts first by entity, then attribute, then value, then tx is called EAVT.An atomic fact in a database, composed of entity/attribute/value/transaction/added. Pronounced like "datum", but pluralizes as datoms.An atomic unit of work in a database. All Datomic writes are transactional, fully serialized, and ACID (Atomic, Consistent, Isolated, and Durable).Indexes store datoms as a tree of segments, where the leaf nodes contain a few thousand datoms each.An atomic fact in a database, composed of entity/attribute/value/transaction/added. Pronounced like "datum", but pluralizes as datoms.An atomic unit of work in a database. All Datomic writes are transactional, fully serialized, and ACID (Atomic, Consistent, Isolated, and Durable).A complete Datomic installation, consisting of storage resources, a primary compute stack, and optional query groups.a CloudFormation stack providing computational resources. Every Datomic system has a single primary compute stack, and may also have multiple query groups.An atomic unit of work in a database. All Datomic writes are transactional, fully serialized, and ACID (Atomic, Consistent, Isolated, and Durable).Sorted collection of datoms. Indexes are named by the order in which datom components are used for sort, e.g. An index that sorts first by entity, then attribute, then value, then tx is called EAVT.The durable elements managed by a Datomic system.The durable elements managed by a Datomic system.a CloudFormation stack providing computational resources. Every Datomic system has a single primary compute stack, and may also have multiple query groups.An AutoScaling Group (ASG) of nodes used to dedicate bandwidth, processing power, and caching to particular jobs. Unlike sharding, query groups never dictate who a client must talk to in order to store or retrieve information. Any node in any group can handle any request.A process that uses a Datomic library to obtain connection to interact with one or more database.Nodes use a multi-layered cache that consists of an object Cache, valcache, and an EFS Cache.Indexes store datoms as a tree of segments, where the leaf nodes contain a few thousand datoms each.An atomic fact in a database, composed of entity/attribute/value/transaction/added. Pronounced like "datum", but pluralizes as datoms.Indexes store datoms as a tree of segments, where the leaf nodes contain a few thousand datoms each.Nodes maintain an on-heap cache of segments containing the most recently used datoms.An SSD-backed cache of segments. Valcache is similar in performance to memcached but durable and capacious.Indexes store datoms as a tree of segments, where the leaf nodes contain a few thousand datoms each.Sorted collection of datoms. Indexes are named by the order in which datom components are used for sort, e.g. An index that sorts first by entity, then attribute, then value, then tx is called EAVT.An SSD-backed cache of segments. Valcache is similar in performance to memcached but durable and capacious.Nodes maintain an on-heap cache of segments containing the most recently used datoms.