Quiz_review

Xingyu Chen

February 23, 2022

Agile: Product backlog, sprint backlog 2-4 week, daily scrum meeting 24 hours, potentially shippable product increment. Agile methods are adaptive rather than predictive. Agile methods are people-orinented rather than process-oriented.

4 Agile values:

- Individuals and interactions: face-to-face conversation. Build projects around motivated individuals.

- Working software: primary measure of progress. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

- Customer Collaboration: Agile processes harness change for the customer's competitive advantage. Satisfy the customer through early and continuous delivery of valuable software.
- Responding to change: At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly (retrospectives)

Extreme Programming: Planning, Managing, Designing, Coding, Testing

Test Driven Development: write a test check if the test fail, write production code.

Continuous Integration: Development Team, source code, version control system, source code build, static code analysis, run antomated unit tests, code coverage analysis, built artefact, setip test fixtures, provision and deploy to test environment, run automated functional tests, publish reports.

Relational database:

Operating on and return tuples (rows), and is independent of the application.

It have no concept of aggregate within the data model, which is aggregate-ignorant (Most people prefer)

Indexing: keep track in query

- Persistence: the most obvious value of a database is keeping large amounts of persistent data

- Concurrency: Relational databases help handle this by controlling all access to their data through transactions
- Integration: A common way to do this is shared database integration where multiple applications store their data in a single database
- Consistent model: developers and database professionals can learn the basic relational model and apply it in many projects.

Database terms:
Row -> tuple : a data set representing a single item
Column -> attribute: a labeled element of a tuple
Table -> relation: a set of tuples sharing the same attributes.
View -> Derived relvar: any set of tuples

ACID:

- Atomicity: Transactions are all or nothing
- Consistency: Only valid data is saved
- Isolation: Transactions do not affect each other
- Durability: Written data will not be lost

SQL: Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language and a data manipulation language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

Object Oriented Database Failure: The role of SQL as an integration mechanism between applications. In this scenario, the database acts as an integration database—with multiple applications, usually developed by separate teams, storing their data in a common database. This improves communication because all the applications are operating on a consistent set of persistent data

Database normalization: Organizing the fields and tables of a relational database to minimize redundancy and dependency. No duplicate data.

First Normal Form (1NF): No repeating elements or groups of elements. Do not repeat your columns.

Second Normal Form (2NF): No partial dependencies on a concatenated key. OrderData and OrderId always same is data duplication.

Third Normal Form (3NF): No dependencies on non-key attributes. CustomerName and CustomerCity into another table and then put a Customer foreign key into Orders.

NoSQL DBs:

- Don't use SQL/Relational Model

- Open source projects
- Run well on clusters
- Tunable Consistency: all node be updated.
- Flexible Schemas
- Resilience to failure
- Handle massive read/write loads
- Self healing

CAP theorem:

Also known as Brewer's theorem, states that it is impossible for a distributed computer system to simultaneously provide all three of the following guarantees:

- Consistency (all nodes see the same data at the same time). All Clients always have the same view of data.
- Availability (a guarantee that every request receives a response about whether it was successful or failed) Each client can always read and write.
- Partition tolerance (the system continues to operate despite arbitrary message loss or failure of part of the system) The system works well despite physical network partitions.

CA: RDBMSs, Aster Data Greenplum **AP**: Dynamo, Cassandra, SimpleDB **CP**: BigTable, MongoDB, Redis

Aggregate:

Allows lists and other record structures to be nested inside it.

Do not have ACID transactions.

However, Customer and order, aggregate will let address record appear three times and ID copied each time.

The clinching reason for aggregate orientation is that it helps greatly with running on a cluster, which is the killer argument for the rise of NoSQL. If we're running on a cluster, we need to minimize how many nodes we need to query when we are gathering data.

Clustering:

The alternative is to use lots of small machines in a cluster. A cluster of small machines can use commodity hardware and ends up being cheaper at these kinds of scales. It can also be more resilient— while individual machine failures are common, the overall cluster can be built to keep going despite such failures, providing high reliability.

- Fault Tolerant: self healing, data replication.
- High Availability
- Eventual consistency

Key-Value data store:

The aggregate is opaque to the database. It is just some big blob of mostly meaningless bits.

Redis:

A in-memory data structure store: supports common data structures such as strings, hashes, lists, sets, and sorted sets. You can think of Redis as a giant dictionary. Highly performant reads

Redis Database File (RDB) Persistence:
Point in time snapshot of your dataset at a specified interval Commonly used for data backups and is very useful in disaster recovery NOT good if you need to minimize the chance of data loss

Append Only File (AOF) Persistence:

Redis can be configured to create a append only change-log that can be used to reconstruct the data store in the event of a failure AOF can be slower than RDB as it has to frequently sync changes with the change-log which can become very large if there are a significant amount of writes.

Persistence is entirely optional with Redis:

Many people use Redis as a feature rich caching system and treat it as an ephemeral data source that can fall down and be reconstructed from a persistent data store ex. Postgres.

Pros:

- Really easy to use
- Highly scalable
- Fast!
- Supports commonly used data structures and nesting
- The Redis community is very active and there are a lot of add-ons/features to Redis that support things like full-text search, pub/sub, geospatial data, keys with limited time-to-live, etc.

Cons:

- Primary key access
- Querying data structures is limited
- Secondary indexes is really only recommended for advance use
- Does not support the rollback of a transactions so making updates of a large data set can be risky
- Since Redis is an in-memory data store persistence must be managed and is not recommended if your system is not tolerant to data loss
- The data structures stored in Redis are incredibly flexible, but that can also make them difficult to deal in code as the data structure may change over time ex. Adding new fields.

MongoDB:
MongoDB stores data in the form of documents, which are JSON-like field and value pairs.

Document database: is able to see a structure in the aggregate. It imposes limits on what we can place in it, defining allowable structures and types. With a document database, we

can submit queries to the database based on the fields in the aggregate, we can retrieve part of the aggregate rather than the whole thing, and database can create indexes based on the contents of the aggregate.

Row-oriented: Each row is an aggregate (for example, customer with the ID of 1234) with column families representing useful chunks of data (profile, order history) within that aggregate.

Column-oriented: Each column family defines a record type (e.g., customer profiles) with rows for each of the records. You then think of a row as the join of records in all column families.

Skinny rows have few columns with the same columns used across the many different rows. In this case, the column family defines a record type, each row is a record, and each column is a field.

A wide row has many columns (perhaps thousands), with rows having very different columns.

A wide column family models a list, with each column being one element in that list.

Cassandra is NoSQL database and good at time series data. node cluster, self-healing, one node dead, another node coming out.

Cloud Server:
Compute Instance, Choose size/image/Permissions and the like, create a template to use the same settings.

Charge for every access but can deploy anywhere.

Terraform: all cloud provider

Ansible: configure all VMs, automation tool for configuring infrastructure and application deployment.

Idempotent: You only do things that are needed and repeatable without side effects.

Declarative (not procedural): you write a description of the state you want and Ansible will execute the steps to accomplish this state.

Product lifecycle management. From Ideation to development to positioning to pricing

Microservice benefits:

- developer productivity
- separation of concerns
- Independently deployable
- Independently Scalable

Microservice complexity:

- More moving pieces
- Logging: expensive due to scale, all events in that process
- Debugging

- Balancing Load
- Multiple/Rolling Deployment
- Monitoring

A container == Linux Process:

- Own process space
- Own network interface
- Can run stuff as root
- Can install packages
- Can run services
- Uses the host kernel
- Can't boot another OS
- Can have it's own kernel modules

Container Benefits:

- Dependencies managed at development time
- Build once, run everywhere
- Lighter than VMs

Orchestration:

Automated deployment is custom code, lifecycle management is custom code.

Version control:

It can track the changes to a file separately from the file itself. We can capture and record snapshots of the file anytime we care to take one.

It gives us a history of the file over time, so you can go back to see what the code was at any point, or roll back to a known good state. Hopefully, too, it can help you find out who did things, and why.

Git:

First, it's a version control system. It tracks the contents of files and changes over time.

Secondly, it's distributed. This means there's no central store or server. You don't have to connect to a server running somewhere to do anything. Every copy is self- contained.

Git uses SHA1 hashes to identify everything.

'git merge' combines the two histories together. From the main branch, we say "merge the new-feature branch". This combines the two histories together, and changes `main` to point at the new merged history.

'git rebase' lets you move the "base" of a branch. Here, we've moved the base of the new-feature branch to be on top of the latest "main".

Github:

Hosting, collaboration, pull request + Branches for everything! + Pull requests for everything! + Dont commit credentials