Seed data is the baseline reference information loaded during initial system setup. It builds the structural foundation by creating core models, default settings, and dimensions like the time calendar. This structural setup happens first, creating the required entities and lookup values before processing regular business activity.

Fact data captures transactional business metrics and quantitative measurements recorded across time. It directly reflects day-to-day operations through metrics like actual sales, customer orders, inventory balances, and demand forecasts. These values sit at specific intersections of master dimensions, such as an item combined with a location and a date. Before this data enters active server memory, the system runs master data checks to catch errors and confirm that all referenced dimension keys actually exist.

—

Integration 3.0 is the system o9 uses to move and clean data. Old systems relied on external software like SSIS to load files, which was slow and hard to maintain. Integration 3.0 runs directly inside o9's cloud using modern tools like Delta Lake, Apache Spark, and Apache Airflow.

Everything begins when a business drops its raw data into cloud storage or an SFTP folder. These files can be sales orders, product lists, or store locations. Integration 3.0 picks up these raw files and loads them into first-stage storage tables called the IN layer. This step keeps the data exactly as it arrived.

Next, the system cleans and standardizes the records using Apache Spark. The data moves into a middle tier known as the ODM layer. Here, the platform fixes messy entries. It drops missing mandatory fields, checks for duplicate rows, and links foreign keys together. Any rows that break business rules are isolated into an error folder on the file server so people can review them.
The clean data then moves into final staging tables called the LSU layer. These tables organize the rows so the o9 planning engine can read them quickly. Integration 3.0 then streams this data straight into the live memory of the GraphCube. Planners can now see updated numbers on their screens without the system freezing or running slow database queries.

All of this work is controlled visually through the o9 web screen. Configurators link steps together on a visual canvas instead of writing complex data scripts by hand. Behind the scenes, the screen turns these visual steps into Airflow workflows that run on a schedule or trigger after another task finishes. Because the jobs run inside lightweight Kubernetes containers, they scale up when data volumes are huge and turn off when the job is done.
—
The Enterprise Knowledge Graph (EKG) serves two main purposes in o9. It acts as a digital map connecting supply chain elements like products, suppliers, warehouses, and customers. At the same time, it provides a simple web screen that builds integration pipelines automatically. Instead of writing complex Spark code by hand, consultants configure data rules using clicks and dropdowns.
Every EKG setup uses two separate tenant environments. The first environment is the Configurator tenant. This is a design space where you select the business project and

choose which tables or fields you need. You also set up quality rules, such as checking for duplicate records or validating parent-child master hierarchies. Inside this space, you map raw incoming files to the IN layer and define join rules between tables. Once finished, you publish the project to generate a setup key.

The second environment is the Deployment tenant. This is the actual working system that runs the data pipelines. You paste the setup key from the first tenant to pull down all the design specs. With a few clicks, the system automatically builds the physical storage tables in Delta Lake. It also creates full sets of Airflow pipelines to move, clean, and stage the data. When the batch runs, source files flow step-by-step from landing storage to final GraphCube memory. Any rows that fail business rules are saved as error files and returned to an error folder for review. If you ever need to apply custom business rules, you can add small Python scripts through built-in exits without breaking the standard platform setup.

—

Config 2.0 changes how o9 stores, updates, and organizes platform models. In the older Config 1.0 approach, all configurations lived directly inside a permanent database on the tenant. If you made changes, you had to back up the entire system using manual database snapshots. Config 2.0 treats tenants as temporary work areas. The true single source of truth is a remote Git repository hosted in Azure DevOps.

The entire platform configuration is split into small pieces called Config Blocks. Base blocks contain standard out-of-the-box features created by product teams. These base blocks remain locked and read-only. When you build customizations for a client, you create a Derived Block that sits on top of the base block. This separate delta layer saves only your changes. Upgrades become straightforward because you can connect to a newer base block without overwriting your custom setup.

Every individual item in the system exists as its own separate JSON file. A single report, a calculation rule, a measure, or a dimension property has its own independent file inside the repository. The platform enforces strict unit boundaries for these items. If two consultants edit the exact same entity at the same time, the system will not try to blend their text lines together. It stops the promotion, flags a conflict, and requires the team to resolve the differences manually.

Day-to-day configuration follows structured Git branches. The Main branch holds the current, stable baseline configuration. Each developer tenant operates on its own isolated Tenant branch. You build and test your adjustments in your tenant branch without disturbing anyone else. When your work is complete and tested, you promote the changes back to Main.

Before a formal software rollout, teams cut a Release branch from Main for acceptance testing. The live Production tenant is never edited directly; it is always updated through a clean reset using the approved Release branch.

When multiple business units need to share metrics across separate blocks, Config 2.0 provides clean connection paths. Master data like dimensions and levels are automatically global and accessible everywhere. Measure metrics stay isolated inside their home blocks.

To pass values between blocks, you can designate an entity as an Interface Measure. You can also configure a Measure Alias in the master solution settings. A Measure Alias points a downstream planning measure to an upstream metric, letting both share the exact same numbers in computer memory without copying data.