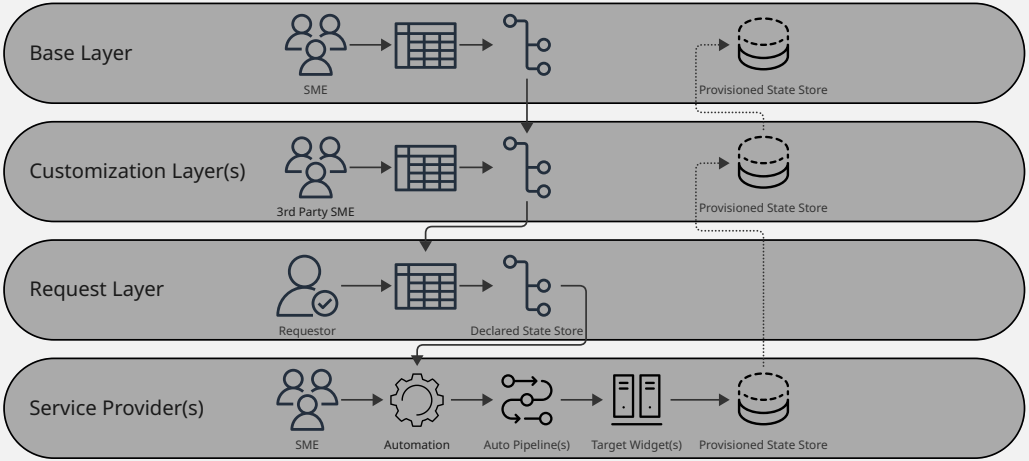

The **objectives** of this data model are to:

- Develop a framework for distributing IaaC/Config data from base standards to consumption pipelines.
- Enable consumption for provisioning and configuring infrastructure and software for production use (e.g., VM provisioning).
- Facilitate drift management, remediation, and monitoring.
- Allow for layered, templated, or direct usage with usage constraints and predictable outcomes.
- Be flexible enough to contain any entity definition and transformable to meet the needs of various consumption tools (Terraform, Ansible, third-party tools).
- Be stored in a Git repository and mimic the Kubernetes YAML format.

The **rules** governing this data model include:

- Host entities are binding entities to which realized entities attach.
- Everything starts with a base entity, and changes are implemented via layer entities.
- Layer entities override previous entities in the precedence chain and must contain and maintain the order of parent entities.
- Base and layer entities are immutable and all entities must be versioned (major, minor, revision).
- Changes to an entity result in a new version.
- Any data can have metadata subtags (e.g., override preference, basis for config value).
- All entities in a chain should maintain the parent format (e.g., YAML).
- Final declared entities should be directly consumable by target operations, stored in source control, and associated with the host entity in a CMDB.
- Each entity should link to its parent and have an origination date/time stamp.

The presentation provides an **example data flow** that consolidates and isolates different layers, including parent repos (standards, base templates, metadata, context, governance, origination date), customization layers eg; dc, network location, dmz, non-dmz. Finally add ing the specific requested widget config resulting in a combine data payload sent for realization by the Service Provider.

A **data model example** highlights centralized configuration and provisioned state, layered/templated customization, "Everything as Code" using standard Git processes, an observable data supply chain, localization at the point of need, data versioning for audit/compatibility, ancillary meta-data, JSON/YAML native format, API native consumption, flexible repo layouts, and a standardized naming schema.