Application Provisioning

While the API allows for traditional storage provisioning, such as volumes and data services, it also enables a modern application-driven provisioning model. With application-driven provisioning, the workload or application view of storage and its associated services are modeled through an application template (AppTemplate).

Some common applications, such as Cassandra, MySQL, Hadoop, VMware environments and test/development workloads, are pre-modeled through default templates. Users may also build custom templates from scratch or by cloning and modifying existing templates.

The AppTemplate is the interface used to express data service objectives on a per-application basis, by configuring application-specific characteristics such as:

- Host access controls (initiators or hosts accessing the data)
- Host targets or exports
- Data volumes
- Performance
- Security
- Data durability & replication requirements
- Data management services, such as snapshots, and their schedule

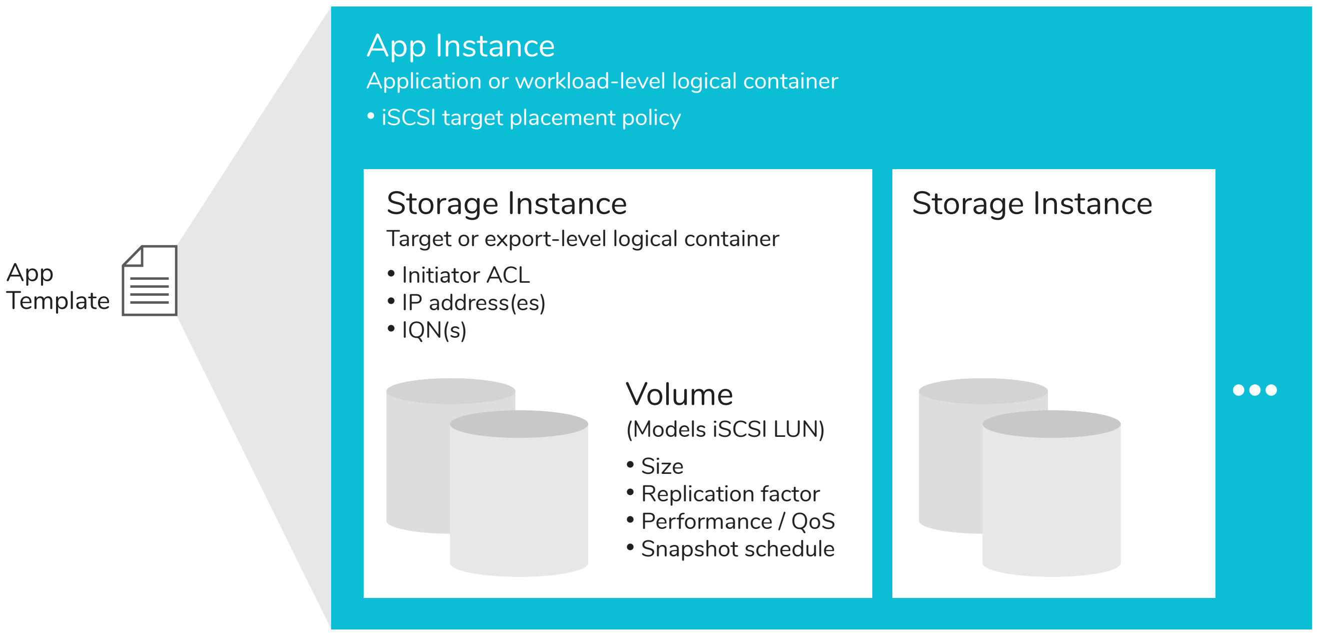

Figure 4: AppTemplate/AppInstance Layout

AppTemplates may be used to instantiate and deploy one or more application instances (AppInstance). An AppInstance is the provisioned entity within the system.

An AppInstance may be associated with a parent AppTemplate, thereby inheriting any configuration/policy changes made to the AppTemplate, or it may be disassociated from the parent AppTemplate.

The application-specific characteristics referenced in the AppTemplate are known as policies. These policies are associated with the AppInstance at the time of instantiation, based on the location of the tenant or user issuing the deployment.

Policies are designed with the flexibility to be inherited or overridden through the template hierarchy based on the requirements. This provides the system with inherent multi-tenancy, policy association and configurability.

Policy Implementation and Multi-tenancy

Let's take an example using Hadoop, MySQL and MongoDB. Each of these applications have different data objectives and requirements. Their objectives may be expressed in terms of performance, consistency model, replication expectations, data protection policy, and so on.

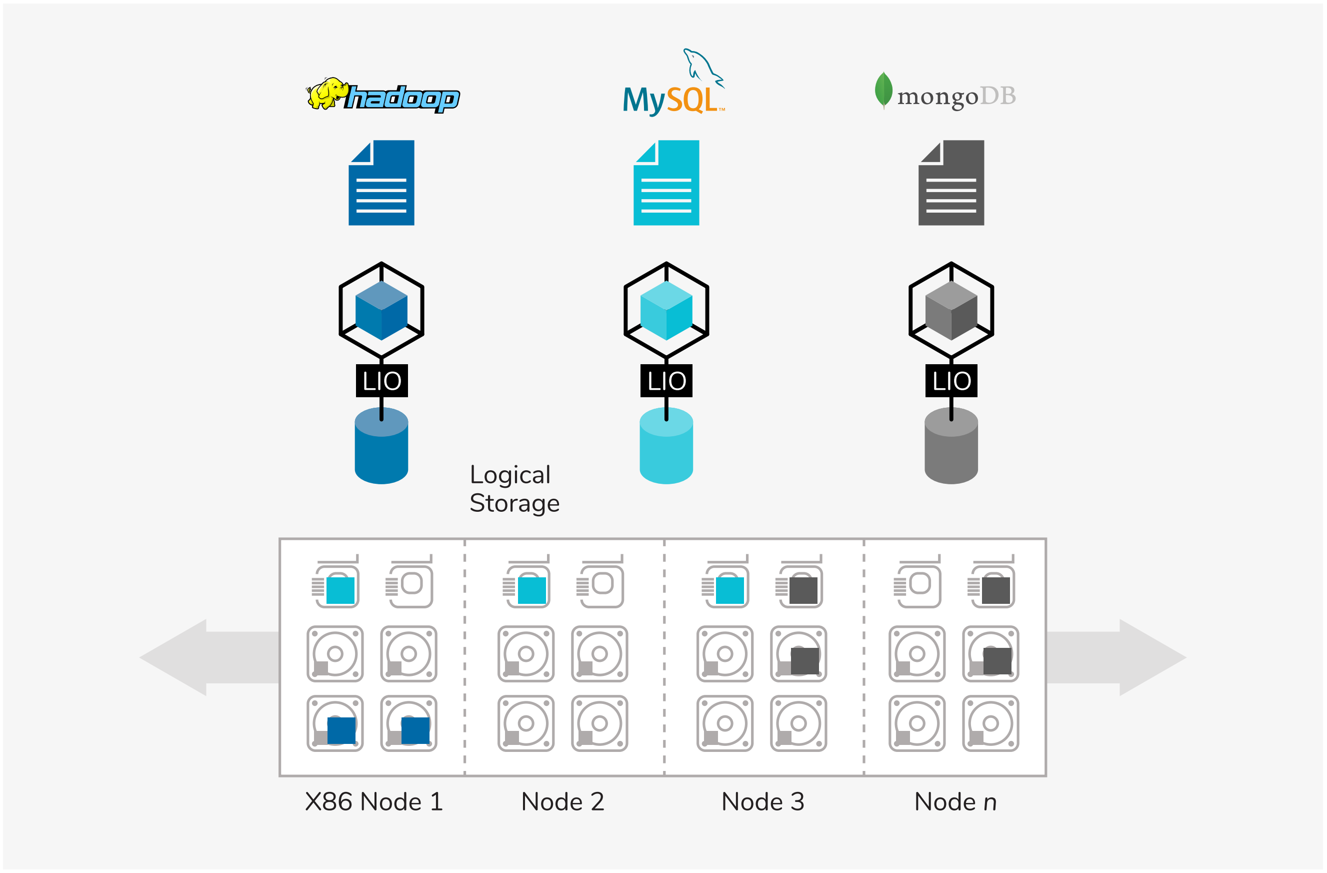

Figure 5: Deploying Multiple Applications based on Policies

An AppTemplate is used to describe the objectives for each application. Each of the polices expressed within an AppTemplate may be inherited or modified based on the tenant instantiating the AppTemplate. The tenant may be a user, an organization or a sub- organization.

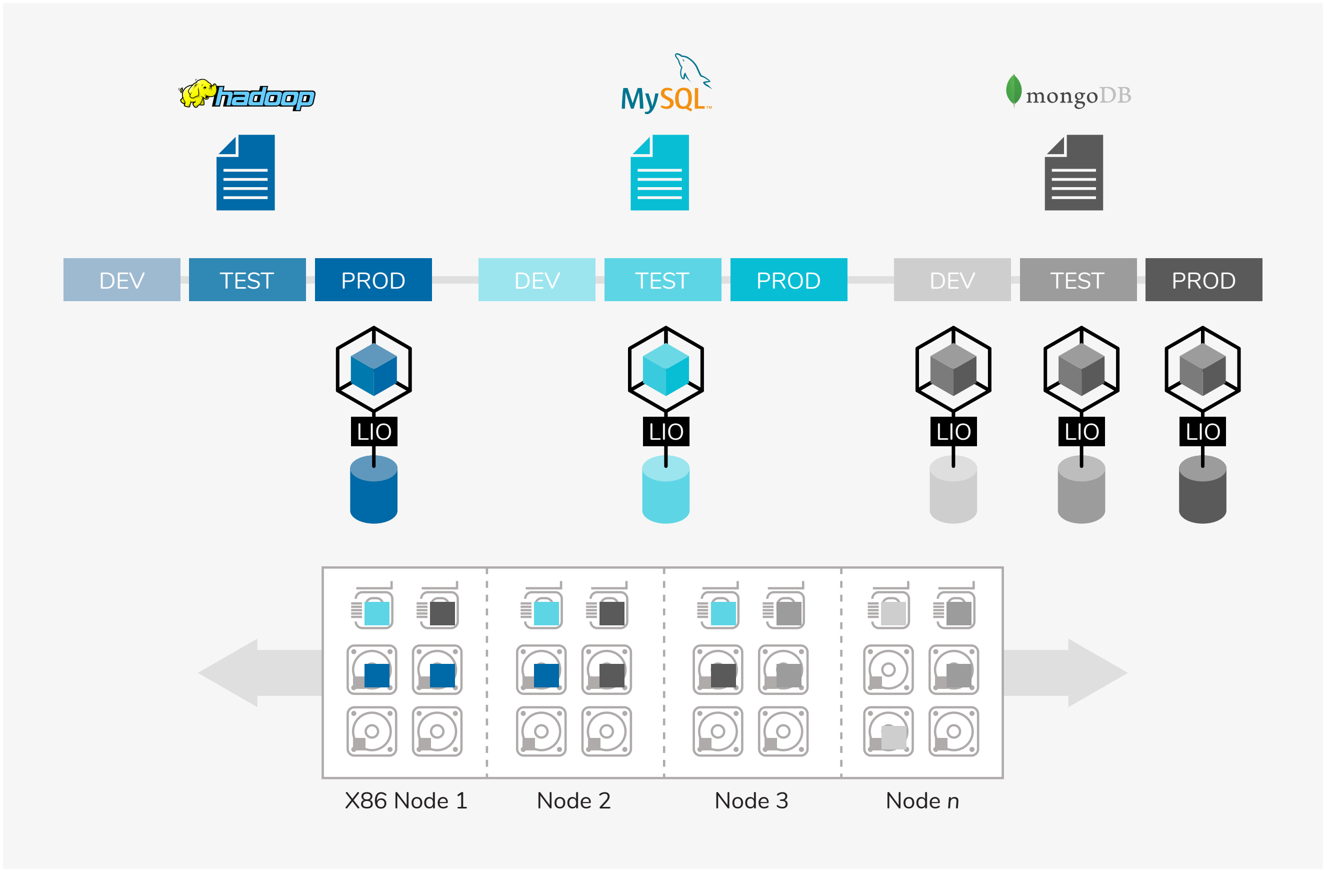

Figure 6: Overlaying Tenant Policies onto Applications

For example, Development, Test and Production tenants might require different policies for the same applications. Therefore, based on the tenant instantiating the AppTemplate, different data services objectives can be overlaid onto an application.

Datera also provides resource isolation and mapping of system resources to the appropriate AppInstance based on its tenancy, policy and associated AppTemplate. This allows for deep multi- tenancy controls across the system, beyond the visualization layer and access control lists for the data volumes.