Dataphoria Cloud Architecture

# Overview



This diagram provides a high-level overview of the proposed physical architecture for the Dataphoria Cloud service. The architecture is intended to provide a physical realization that can be initially implemented as a bare-bones cloud-capable offering, while at the same time support scaling out at any given point to support increased loads.

Note that with the components as currently specified, this architecture would not support per-instance scaling, but that could be addressed by providing a way to synchronize and/or distribute instance data. Note also that it does not support the ability to scale out the data layer, that functionality would require some form of data partitioning or other scale out solution. This architecture does not preclude the addition of either of those scaling points at a future time.

# Data Cluster

The Data Cluster component will be implemented as a simple fail-over cluster using PostgreSQL instances deployed to Ubuntu server nodes.

# SAN Storage Cluster

This component should be completely independent of the D4 service farm, each service node would just need to be able to access the instance folders stored here. The only outstanding question here is whether the SQLite device could attach to a data file stored on a network or SAN-connected device. This should be possible, but it is a risk to the architecture defined here.

# D4 Service Farm

This component is a collection of N nodes running Ubuntu server with the Dataphor and Dataphor Listener services installed. The Listener component of Dataphor will be extended to provide instance management functionality.

## IInstancer

The IInstancer API provides the following methods:

* **Create** – Creates a new instance with the given server configuration information.
* **Delete** – Delete the given instance
* **Start** – Starts the given instance
* **Stop** – Stops the given instance
* **Kill** – Kills the given instance
* **Ping** – Pings the given instance to determine health

# Coordinator Cluster

This component coordinates instance management for the service farm. Nodes can be added and removed from the cluster, instances can be requested and removed, and usage data can be retrieved and reported.

Note that this component could be deployed as a singleton service. This would be a single point-of-failure for the system, so it should be cluster-enabled at some point, but the minimum viable product does not necessarily need clustering for this component.

The coordinator component accesses the Listener and Instancer services running on each node in order to perform actual instance management.

## ICoordinator

The ICoordinator API provides the following methods:

* **RegisterInstance** – Registers the given instance as being present on a given node.
* **UnregisterInstance** – Removes the given instance from a given node.
* **EnumerateNodes** – Returns the set of nodes in the farm.
* **EnumerateInstances** – Returns the set of instances in the farm.
* **RegisterNode** – Registers a new node in the farm.
* **UnregisterNode** – Unregisters a node in the farm.
* **DeployInstance** – Deploys an instance to the given node of the farm.
* **RemoveInstance** – Removes an instance from the given node of the farm.
* **StartInstance** – Starts the given instance.
* **StopInstance** – Stops the given instance.
* **KillInstance** – Kills the given instance.
* **PingInstance** – Pings the given instance.
* **ReportUsage** – Reports usage information for instances in the cluster.
* **RetrieveUsage** – Retrieves usage information for the cluster.

# Web Service Farm

This component is a traditional scaled out stateless web site running Apache w/ mod-mono on Ubuntu servers. This component would access the coordinator to determine how each request to the cloud should be routed.

