MathOps.Dev Server Design

The MathOps system is designed to be deployed on a bank of servers that provide availability in the event of any one server failing, and to support live updates of production applications without downtime. This allows individual servers to be patched and upgraded without downtime.

Network Design

Servers must be accessible from the public Internet on HTTP (80) and HTTPS (443). In addition, if a local testing center is operating, servers meed to be accessible from the testing center on ports HTTP (80) and HTTPS (443). Each server should have two NICs, one for the public network, and one for the private network.

All servers will run **keepalived**, listening on a common “well-known” IP address on the public network, while interconnecting with each other on the private network. This daemon will also monitor the “httpd” process for failures and fail-over to backup servers when the primary httpd front-end goes down.

All servers will run **glusterFS** over the private network to share a directory structure that includes static web pages, images, closed-caption files, etc. One machine will be the primary server, and will distribute file updates to clients. If the primary goes down, the clients have copies to use in the interim, but could not accept automated updates until the primary is running again (or until some other machine is made primary).

Production and test servers will run an instance of **Apache httpd** with TLS certificates and a **Shibboleth** service provider, listening on the “well-known” IP address (ports 80 and 443) on the primary network, forwarding requests to an HTTP/AJP server via proxy (**Tomcat**, for example). Development servers can simply run the HTTP server.

All servers can serve static files from the glusterFS filesystem, or can forward service requests to any of the registered web services. These web servers are stateless – any server can serve any request. Keepalived could route all traffic to the primary and keep the others as hot backups, or could load-balance traffic to all servers.

Shared Data Services

Since any server could be asked to field any request, state must be shared across servers. This includes long-term state such as database tables, as well as short-term state like login sessions or cached query data.

All servers will run an instance of the **Apache Cassandra** replicated database, with replication factors that allow at least one (and preferably more than one) server to fail without loss of data or service. This will store long-term data. This container will only have access to the private network. Any web service may access long-term data from this data store.

All servers will run an instance of the **Redis** in-memory key-value database, with replication factors that allow at least one (and preferably more than one) server to fail without loss of data or service. This will store short-term data like login sessions, assessment sessions, and cached user-specific data queried to serve requests. This container will only have access to the private network. Any web service may access short-term data through this data store.

Web Service Proxy

Each server runs a web service proxy, which can determine the hostname and URL path to which a request was directed, and can forward requests to services as needed.

The proxy listens on the **localhost** network for the registration of new services (these registrations map a hostname and path prefix to a target service, specified by a TCP port on the **localhost** network).

When a request arrives, the proxy looks up the associated service, and creates (or retrieves from a pool) a connection to that service, and forwards the request. When the response arrives, it is sent to the client; if the respopnse times out, or the connection is broken, an error is returned to the client. If the host/path combination does not map to any registered service, a NOT FOUND response is sent to the client.

When a service starts and registers itself with the proxy, there may already be a service registered with the proxy for that host and path prefix. Services can tell the proxy to either stop sending requests to the old service and to start sending all new requests to the new service instance, and to tell the old instance it can terminate once the last session is completed. They can also tell the proxy to load-balance among the multiple service instances.

The proxy will support visibility and management through a reserved path prefix on each host (this requires a login) and can examine the set of all registered services, retrieve statistics, view error logs, and disable registered services. It could even have the ability to start new service instances if services are deployed with a standard structure.

The proxy can be configured to serve static files under specified paths, where a path is mapped to a directory.

The proxy is designed so that if it needs to terminate, it can persist its service registrations to a file, and when it starts, it can load that file to restore state. The proxy stores no other state. Restarting the proxy will not cause any client disruption unless a request arrives in the short interval when the proxy is restarting.

The web service proxy lives under the "dev.mathops.web.proxy" package.

Web Services

Every web site, web service, or websocket service provided within the system will be deployed as a separate service. Each will listen on its own TCP port, to which the front-end will forward traffic by reverse proxy to serve requests. These services will only be accessible from the private network.

The **WeBWorK** web services requires its own MySQL database. This can either be deployed on a single server, or MySQL replication can be set up to allow it to run across all servers.

In addition commercial products can also be deployed on distinct TCP ports and served through the front-end. For example:

* **Mantis** issue tracking
* **Grafana** log aggregation
* **Prometheus** metrics collection
* **Nagios** monitoring
* **TestContainers** integration test automation
* **Flow.CI** continuous integration