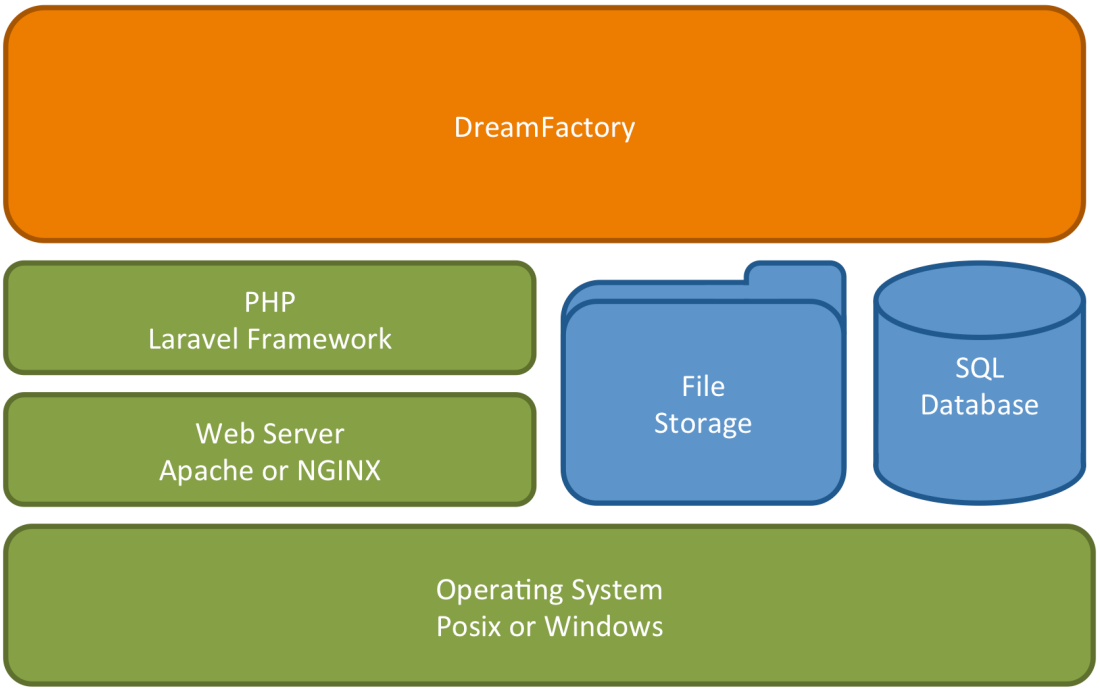
# DreamFactory Architecture Guide

This white paper is designed to provide architecture information about the DreamFactory instance. The sections below discuss the various components and characteristics of the system and an anatomy of various API calls as they travel through the system.

## Basic System Architecture




DreamFactory is an open source REST API backend that provides RESTful services for building mobile, web, and IoT applications. In technical terms, DreamFactory is a runtime application that runs on a web server similar to a website running on a traditional LAMP server.

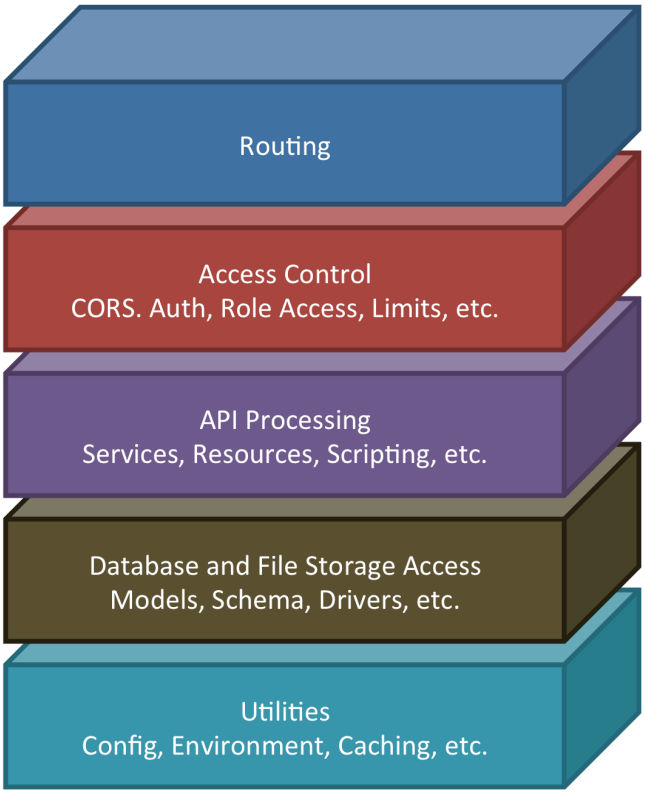
In fact, as a base, we require a hosting web server like Apache, NGINX, or IIS. DreamFactory is written in PHP and requires access to a default SQL database for saving configuration. Depending on configuration for caching, etc. it may or may not need access to the file system for local storage. If pre- and/or post-process scripting is desired, access to V8Js or Node.Js may also be required. It runs on most Linux distributions (Ubuntu, Red Hat, CentOS, etc.), Apple Mac OS X, and Microsoft Windows.

Installation options are highly flexible. You can install DreamFactory on your IaaS cloud, PaaS provider, as a Docker container, on premises server, or a laptop.

Installer packages are available, or the DreamFactory source code is available under the Apache License at GitHub.

**DreamFactory Components**

The DreamFactory application can logically be divided into several operational components. While these components are logical and do not necessarily represent the code structure itself, they are useful in discussing the subsystems and functionality of the application, as well as the anatomy of the API call later on.




**Routing**

Routing sets up the supported HTTP interfaces to the system. In tandem with Controllers, this component controls the flow of the calls through the system. Controllers are essentially groups of routes or HTTP calls that share some logical handling and are paired with a set of access control components. There are essentially three controllers that the routing component can hand the call off to.

- Splash Controller - This handles the initial load and setup states of the system. It also routes web traffic to the default web application (i.e. Launchpad), where users can login and access other configured applications like the admin console, etc.

- Storage Controller - This handles direct file access to any file services where folders have been made public through configuration. Files are requested via the service name and the full file path relative to that service. The file contents are returned directly to the client. This is primarily used for running applications hosted on the DreamFactory instance.

- REST Controller - This is the main controller for the API, it handles the versioning of the API and routing to the various installed services via a Service Handler. It also handles any system exceptions and response formatting. The Service Handler communicates generically with all services through a service request and response object.

**Access Control**

Access Control is made up of middleware, groups of checks and balances that can be used to control access to various parts of the application. The services and resources for Access Control consist of the following:

- System status checks
- Cross-Origin Resource Sharing (CORS) configuration allowances
- Authentication via user login, API keys, and/or session tokens
- Authorization via assigned user and app role access
- And usage limit tracking and restrictions

If any of these checks fail, the call is denied and the correct error response is sent back to the client; no further processing is done. If all of these checks pass, the call is allowed to continue on to one of the handling controllers, which routes the call for the appropriate API processing.

**API Processing**

At this point the API can be broken down further into logical components that we call Services. Services can be anything from a system configuration handler (i.e. the "system" service), to a database access point, or a remote web service. Service types can be dynamically added to the system to expand service offerings, but many are included out of the box and are list here.

- **System Management**
- **Swagger API Docs**
- **User Authentication**
    - Standard Auth (username/password)
    - Standard LDAP
    - Active Directory LDAP
    - Facebook OAuth
    - GitHub OAuth
    - Google OAuth
    - Twitter OAuth
- **SQL Databases**
    - MySQL
    - SQL Server
    - PostgreSQL
    - Oracle
    - IBM DB2
    - SQLite
- **NoSQL Databases**
    - AWS DynamoDB
    - Azure Tables
    - CouchDB
    - MongoDB
    - SalesforceDB
- **File/Blob Storage**
    - Local File Storage
    - AWS S3
    - Azure Blob Storage
    - OpenStack Object Storage
    - Rackspace Cloud Files
- **Email (for sending email only)**
    - Local Email
    - SMTP Email
    - AWS SES
    - Mailgun Email
    - Mandrill Email
- **Notifications**
    - AWS SNS
- **Remote Web Service**
- **Custom Scripting Service**

Once API calls pass the access controls, they are then handled by the controllers and routed based on the service name given in the call to a Service designated to handle that call.

All calls to the DreamFactory API take the form:

<verb> <http[s]>://<server_name>/api/v2/[<service_name>/[resource_path]]?[<parameters>]

- The HTTP verb (POST, GET, etc.)
- Server name
- API designator
- Version indicator (v2)
- Service name for routing
- Resource path (if available)
- Query parameters and headers

Services may be broken down into logical components called Resources that handle any subtending endpoints from the main service endpoint. For example, in a call to **https://example.com/api/v2/system/environment**, the service is "system" and the resource is "environment".

A majority of the processing done on the server is handled by Services. Services and Resources are both built on the concept of a REST handler. Therefore, a Service may process the full request directly, or choose to pass along the request or parts of the request to any number of resources to process the call based on the path given.

**Server-side Scripting**

Part of this REST handling by the services includes server-side scripting. Each API endpoint, be it a Service endpoint, or a subtending Resource endpoint, triggers two processing events, one for pre-process and one for post-process. Each event can be scripted to alter the request (pre) or response (post), perform extra logic including additional calls to other services on the instance or external calls, as well as halt execution and throw exceptions. Scripting can be used for formula fields, field validations, workflow triggers, access control, custom services, and usage limits. The role-based access controls have separate settings that govern data access for both client-side applications and server-side scripts. This capability enables server-side scripts to safely perform special operations that are not available from the client-side REST API.

The event scripting all happens in the context of the original API call. Therefore, event scripts block further execution of the API call until finished.

DreamFactory uses the V8 Engine developed by Google to run server-side code written in JavaScript. The V8 engine is sandboxed, so server side scripts cannot interfere with other system operations or resources.

In 2.0, DreamFactory also provides access to use Node.js and PHP as a server-side scripting environment. These environments are not sandboxed however and care must be taken when used.

**Database and File Storage Access**

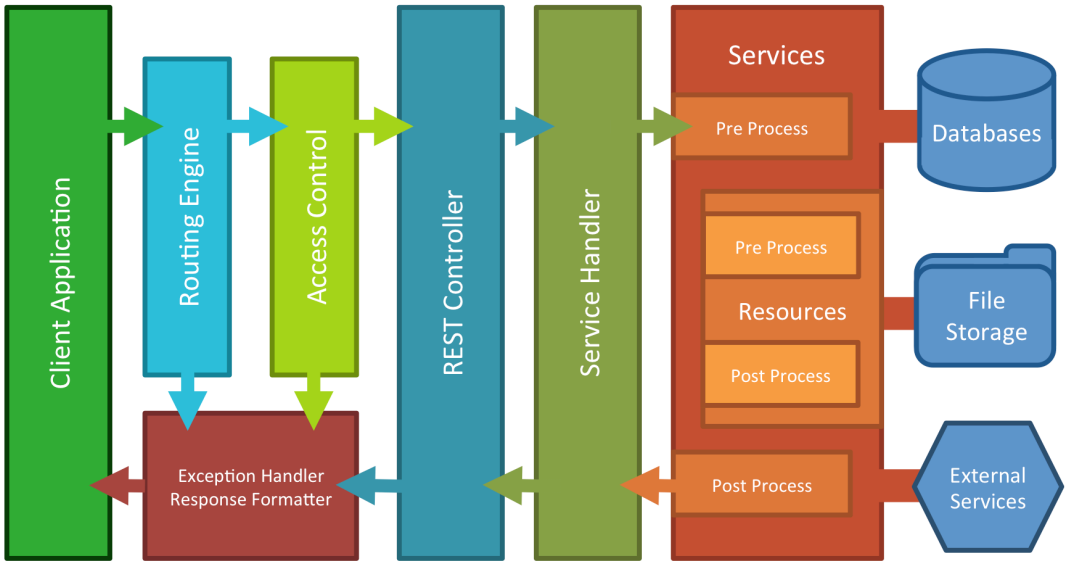
Many of the services mentioned above eventually need to access some data or file store or communicate with a remote process or server. DreamFactory takes advantage of many available open-source packages, SDKs and compiled drivers to access these other resources.

In the case of database accesses, DreamFactory utilizes PDO drivers for SQL databases, as well as, other compiled drivers and extensions like MongoDB, and even HTTP wrapper classes for databases like CouchDB that provide a HTTP interface.

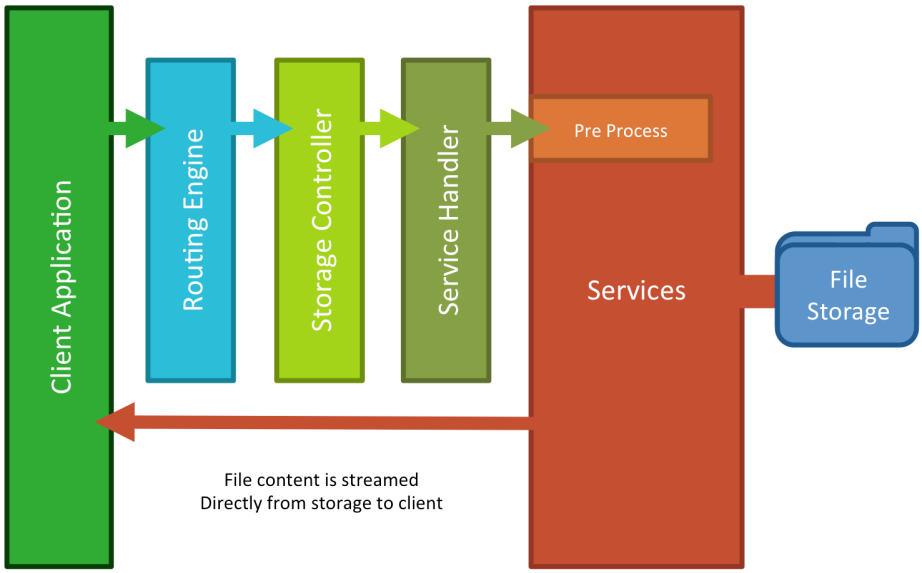
DreamFactory provides internal models and schema access for frequently used data components, particularly the system configuration components. The most frequently used are also cached to reduce database transactions.

A DreamFactory instance may utilize local file storage or various other storage options such as cloud-based storage. DreamFactory utilizes generic file access utilities that support a majority of the storage options, thus giving the system, and thus the API, a consistent way to access file storage.

## Anatomy of an API Call




## Anatomy of a Storage Call



## In Conclusion

DreamFactory is designed to be secure, simple to use, easy to customize, and dynamically expandable to meet most of your API needs. Reach out to the DreamFactory engineering team if you have additional questions of concerns.

**Community Forum**

**Bugs and Feature Requests**

**Contact Support**