Project Abstraction Layer (PAL)

Project Definition

A unified open source initiative that takes a normal piece of web-based software and transforms it into a Software as a Service (SaaS) provider.

Revision History

Author	Date	Description
James Biviano	17/09/2019	Initial document creation
James Biviano	25/09/2019	Document creation
James Biviano	03/10/2019	Document creation
James Biviano	28/10/2019	Adding multi-Application support
James Biviano	02/11/2019	Corrections
James Biviano	04/11/2019	Corrections and additions
James Biviano	06/11/2019	Additions to Database Abstraction
James Biviano	22/11/2019	Additions to Filesystem Abstraction

Preface

Cloud Computing, or more specifically SaaS within Cloud Computing, amasses many topics ranging from its philosophy to its implementation. As such, SaaS has been one of the most controversial branches of Cloud Computing since its inception from the early-to-mid 2000's and boasts one of the broadest topics in computing today.

SaaS's concerns include: methodology; security principles; deployment strategy; data retention strategy and policy; service level management; and mitigation practices. Furthermore, SaaS's market segment alone overshadows both Infrastructure as a Service (LaaS) and Platform as a Service (PaaS) markets combined. In 2019 the SaaS Cloud application market will reach $US113.1 billion annually, with the expected growth to reach $US185.8 billion annually by 2024.

Project Abstraction Layer (PAL) aims to provide an approach or series of approaches that will allow solution providers the ability to use any normal piece of web-based software found in your typical GNU/Linux / UNIX based web application ecosystem and turn it into a true Software as a Service provider platform. This document serves as to provide the initial planning stages of PAL in the form of a functional specification.

This functional specification is divided into independent sections that will make up the entirety of the project. Each section will come with its own summary and then will extend into various implementation strategies or extended feature requirements. The functional specification will then lend from these various sections, making up initial planning stages.

PAL aims to be open in its transparency and open in its source creation. Meaning that contributions and counter-arguments to prescribed methodologies are openly welcomed. Contributors are also welcome to aid in the development of the project. PAL aims to provide a solution for small-to-large businesses, understanding that such an undertaking may take time to refine and perfect.

Tables of contents

What is Software as a Service

Software as a Service (SaaS) is a software delivery approach that occupies a series of rules which have been defined since the early-to-mid 2000's. To clarify these rules, and place them within a more contemporary context, below is a definition of what SaaS is within the scope of PAL.

Technical definition

Application: A piece of executable software.

In the case of SaaS, executable software usually refers to a series of web service-based scripts deployed by a web server, or what's typically considered the mechanisms which make up interactive websites. In modern terms this is more commonly referred to as a Web Application and, in even more evolved terms, a Progressive Web Application. However, within the scope of this document to, encapsulate the entire web site / web application market, we will use the term "Application".

It is important to note that SaaS does not have to be bound to just web-based applications, as SaaS could be based on email or other distributed services. However, within the scope of PAL, it will be applied to only web applications and other web related services.

Instance: An iteration of the application being served which relates to a unique identifier which in this case will be referred to as, "Instance".

An Instance usually caters to an entity such as a company, but ideally an Instance can be anything. For example, a department of a company uses one Instance and another department uses another Instance; an individual can even be seen as an Instance, for example, a personal blog.

Module: A segment of an Application that has been divided to serve as separate functional roles.

In the case of PAL, modules that are within the Application, we give the definition "Instance modulation".

Instance modulation gives the developer the ability to create extended functionality within the Application. For example, one module for shopping carts, another module for photo galleries, and another module for reservations and so on. The term "Instance Module" describes a functional module per Instance scenario and what can be enabled for one Instance may not be enabled for other Instances.

This concept gives SaaS a very powerful basin to develop value-adds. This is because an Instance, instead of just having just one piece of software that a provider can sell the provider can develop many different solutions in varied formats, mix and combine modules and provide extended features to a pre-existing customer base, or use those features to attract new customers.

Integrator: A technical person who makes customisations to Instances.

An Integrator is usually responsible for linking third party services, applying look-and-feel modifications, or making general customisations which follow the SaaS provider's charter or references. Integrators are usually web developers, hosting (re)sellers, or anyone within the web services arena.

What defines SaaS

1. All Instances are to run on or within the same Application

Whatever end file, or series of files, that has been produced to power one Instance of the application, must be used to drive all Instances asynchronously. Furthermore, it is quite possible to confuse this terminology to mean something slightly different. That is, storing cascading versions of the application within each Instance environment. Which is then synchronized by some form of management software and is simply deployed.

SaaS is application driven, if the second principle starts to creep in, the solution begins to fall away from any of the prescribed definitions of SaaS and becomes something slightly different.

2. On-demand provisioning

What this essentially means is that resources are provided at the flick of a switch or by adjusting variables, for example, just at the end of a signup script a new Instance is created. Automated provisioning is considered on-demand, however, doesn't define the term, in the case that resources can be manually administered per Instance still qualifies the term on-demand.

On-demand is a technical definition which describes the activation process of an Instance, or the allocation of resources via per Instance based licensing constraints (see chapter viii. for customer licensing). The opposite to this concept usually relies on complicated software implementations and administration processes to ensure management and consolidation of cascading installations per Instance. This entails many disk writes and installation procedures which on-demand should never really entail, and if it does, should be kept to a minimum.

In the case of PAL the Instant Management Portal will aim to accommodate Instance configuration, creation and management in an on-demand fashion and depending upon configuration may or may not have automated provisioning, however, the solution will still be classified as "on-demand" because of the activation process.

3. Data is stored in a multi-tenant environment

Visualize the concept of a multi-tenant environment as you would a park full of shared storage containers or an apartment building of occupants. In Cloud Computing it is the same but with your data. Data is shared in the same area or environment as all other Instances. Instead of the counter, which is that each Instance uses a unique service to manage its data independently.

PAL will talk about the specific and varied approaches to databases in one chapter. As for flat file data storage this will be discussed in a subsequent chapter just about Filesystems.

4. Per use or consumption-based licensing

Does per use licensing deduce that the product is SaaS? If your business model relies on commission from a sale based on a SaaS offering then the per use license or volume licensing is not applied. Therefore, in some ways it may dissuade it from being SaaS. I would like to say no, but some may say yes.

PAL intends to provide a solution that deals with Instance automation and the customer sign-up processes with a series of APIs. PAL will also intend to facilitate Application consumption and use and provide reasonable data to allow businesses to tailor their own cost models independently.

Simply put, Software as a Service is a piece of software which is provided on-demand which uses the same application (usually a web application) for all its customers, the application stores its customer data in a multi-tenant shared shortage solution and is sold to the customer with licensing based on consumption.

General Considerations

The core purpose of writing this document is to create a functional specification. The functional specification has a few considerations which should be outlined before any form of implementation takes place.

Selecting the correct operating system

Industry strength is one of the key considerations as it would be ideal to support most operating systems. For the sake of portability across most ecosystems the initial development will be focused around GNU/Linux relying strongly on standards such as POSIX, BSD and System V with the mindset to accept more platforms as development evolves.

Solution anatomy

The functional specification has been divided into various separate sections and roles. These sections could be designed monolithically as part of a single system encapsulated within the web server solution. Alternatively, PAL could have some of its sections written as separate services or daemons and have other sections treated as sub sections within other systems. For example, Database and Filesystem Abstraction could be treated as separate services while Host Management is a module inside the Application Environment.

It is important to note that regardless of how the first four (4) sections of PAL are built, Instance Management (see chapter vii.), which is inherently PAL's web based administration portal, will have to be a treated as separate and developed using a web scripting framework which will interact with services based on permissions granted by the operating system similar to how a web hosting control panel would typically operate.

Configuration design

When dealing with configuration, the concept of database-based configuration will be one of the most preferred methods of storage, especially when dealing with Instance configuration.

Other configuration aspects that deal with abstracted services such as Host Management (see chapter iii) and even the Application Environment configurations (see chapter iv) could be flat file or database depending upon what is deemed most feasible for each section that makes up PAL.

Broadness approach

PAL needs to be a solution designed around the principle that not all methods are made equal and therefore must prescribe to doing things in more than one particular way or approach. PAL needs to be abstract in the sense that if there is a new method that can be applied then the solution should have provisions to either allow for that, or to allow for it to be implemented in the future, without effecting other services which operate in connected areas. This approach will be critical for how various sections within the entire solution should work.

A typical example of how this approach will work is irrespective of what Database Abstraction method or methods are used the Filesystem Abstraction method or methods will not be affected by design. If a new Database Abstraction method is added this makes no interruption to other services, unless the new method creates an exception or condition.

PAL aims to offer this approach with the focus of offering industry resilient components such as modules or independent services or daemons that could also be useful for other related projects.

Selecting the correct programming languages

Initially, when reaching for this project, the first thought was whether to use existing software solutions and build modules to cater to PAL's development requirements, or to use the best and newest frameworks and programming languages on the market today.

Since Rust has been spoken about in respect to security and efficiency as being the premiere choice for new projects. Rust could possibly be the choice for PAL's new software implementations.

However, Projects that pre-exist that can be forked (see https://en.wikipedia.org/wiki/Fork_(software_development)) or contributed to directly, irrespective of language, with the aim to service as a component of PAL will be considered as well.

Pre-existing solutions and modularisation

Since web services have such a tremendous supply of open and closed solutions in the market, PAL is open to implementing pre-existing solutions and augmenting those solutions via the means of modulation (See https://en.wikipedia.org/wiki/Modulation) or, if necessary, forking preexisting repositories. The chosen web server solution that will be used as the Application Environment and/or Host Management sections will perhaps be one of the biggest considerations.

In many areas solutions such as Apache offer all the latest and greatest features, such as a highly configurable web server platform that supports pretty much all known language types. Apaches' modulation features and Apache's Portability Runtime library (https://ci.apache.org/projects/httpd/trunk/doxygen/group__APR.html) make Apache quite an acceptable solution that has been refined and industry tested since the dawn of the World Wide Web (WWW).

Apache also works with backwards compatibility and legacy services and is always being benchmarked as the web server of choice in countless scenarios. Apache also supports most scripting language types making Apache a very powerful service to begin with. In this case, custom module creation for Apache vs creating a new service or daemon (https://en.wikipedia.org/wiki/Daemon_(computing)) vs using preexisting web service solutions, is perhaps one of the biggest points of concern for this project moving forward.

Data redundancy and networked services

Unfortunately, the scope of PAL will not cater to these particular areas at this point. These sorts of areas will come down to the individual administrator and server configuration and usually fall in line with custom enterprise solutions anyway. I have decided to keep this area out of scope for the time being. If the project merits expansion then a second functional specification can be created around these requirements.

Host management

Summary

When a visitor accesses a website address or Universal Resource Location (URL) via a web service, the Host Management section will need to match the website address against a database of possible Instances, thus tying the Instance, Application, Module and User to one interactive session.

Since every website is different PAL aims to provide the best approach at integrating solutions into limitless web delivery scenarios. This is achieved by tying any preconceived URL combination into a per Instance Application binding, which makes up the overall expected functionality of the Host Management solution. For example, http(s)://ANYTHING/MAYBE_ANYTHING/[the_applications_relative_elements].

Furthermore, this means that Secure Socket Layer (SSL) certificates will need to be linked to each URL definition. A *many-to-one* relationship as a bare minimum for *many* associated URL host entries to *one* SSL certificate entry will be needed as to allow for Wildcard SSLs (https://en.wikipedia.org/wiki/Wildcard_certificate). As the aim is to provide a limitless URL referencing tool to fall in line with an efficient and elegant approach for Host Management the following URL structures statements should be considered.

Terminology

Saascompany = The SaaS provider
Instance = Reference to the Instance and Application
Module = Reference to the module within the Application

Example URL combinations

https://www.saascompany.com/instance/
https://instance.saascompany.com/
https://www.instance.com

Consider more complex breakdowns ..

https://www.saascompany.com/instance
https://instance.saascompany.com/module
https://www.instance.com/module
https://module.instance.com/
https://moduleinstance.com

Consider language friendly breakdowns ...

https://en.saascompany.com/instance/module
https://pt.saascompany.com/instance/module
https://instance.saascompany.com/en/module
https://instance.saascompany.com/pt/module
https://en.instance.com/module
https://pt.instance.com/module

https://module.instance.com/en
https://module.instance.com/pt

and so on…

Host configuration

Since the solution needs to satisfy on-demand provisioning and access to configuration constants will require multiple disk accesses, using a database is the most feasible option for Instance configuration storage.

The basic topology of Instance configuration for Host Management will be based on a series entries called host entries, which will encapsulate the following articles of data:

- Application Environment identifier
- Instance constraints
- Host data or the URL
- Directory targeting
- Instance modulation constraints
- User management and permissions
- Other possible configuration constraints

Once a web server request is received, or someone visits a URL, this request needs to be matched against a database of host entries, the correct entry will then be used to supply information to other sections within PAL during the visitor's session.

URL generation and management

In further respect to on-demand provisioning (see chapter i), Host Management will also need the ability to auto generate URLs dynamically based on some form configuration format. For example, Host Management is configured to generate a new subdomain at the time of Instance creation using the following scheme, https://{INSTANCENAME}.domainname.com/ where INSTANCENAME is an identifier for a company or username specified during setup.

Further to this ability the administrator, integrator, or even user should have the ability to add or remove custom URLs in an ad hoc fashion, without any restrictions on domain format meaning entire domain name, sub-domains or URLs with directory paths should be easily administrable.

Configuration caching and service efficiency

Since, host entries will initially come from a storage medium it would be a preferred concept to offer a caching mechanism that stores this data within Host Management's memory.

For Instance, having constant SQL requests, or even just flat file requests, every time a web server request is made will cause overall performance overhead, therefore, as many optimisations that can be made programmatically to reduce storage medium requests should be employed.

Instance modulation

With the principle of Instance modulation, the solution should house functionality to specify a *moduleid* along with a suffix for a relative file path or an absolute file path to the modules' destination.

Moduleid should not be compulsory or be bound to the term 'module' either. The interface described in this specification should accommodate any id convention or combination to be used in this manner. Even the naming convention of *applicationid* could be used, or better yet being able to pass composite references for Instance modulation should be possible as well.

From a functional perspective having the flexibility to create a configuration tool that allows custom ids and other variables to be passed to the Application Environments' Inter-process communication mechanisms would be ideal (see chapter iv).

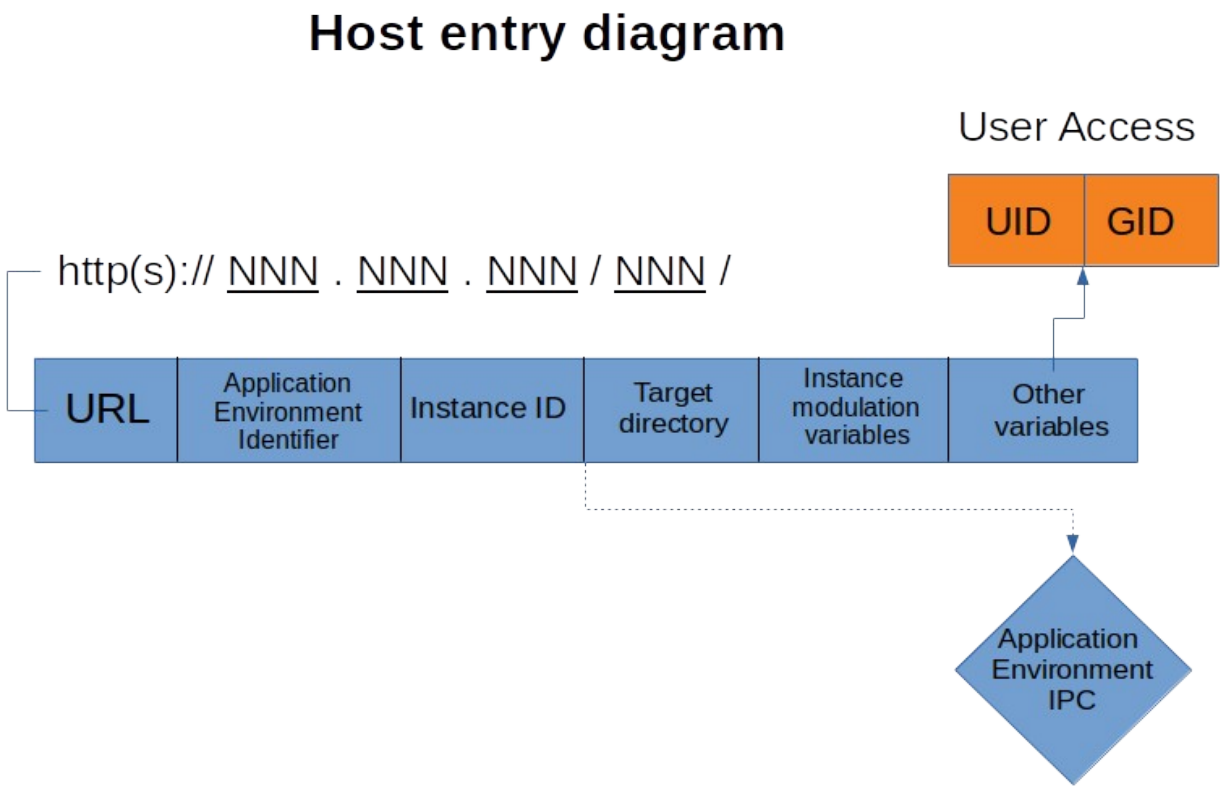
User and group management

Host Management should give web based authorisation protocols a chance to authenticate the visitor before allowing access to certain URLs. Host Management should then deny a user if authentication has not been granted. An example of this is when the customer visits their Administration portal, Host Management should have the ability to redirect an unauthorised user to a neighboring URL until the Application authorises the visitors' session. This means that the visitor should be unable to access entire regions of the Application until authorisation is met.

Ideally all users should be authenticated as anonymous initially and have low level access credentials applied with the Application or the Application Environment being responsible for upgrading this access level. User and group management of this nature is only created as a basic provision, ideally within PAL the aim will be to later use those user and group credentials with other sections within the system.

Graphical representation

Please see the following diagram.



Functional specification

Please see the following articles.

#	Description	Class
i.	The solution will make up part of a web service or daemon as part of a module	Necessary
ii.	The solution will select the correct host entry from Instance configuration, based on the URL combination supplied by the visiting user.	
iii.	Once the correct host entry has been found the web server will need to access the chose Application Environment (see chapter iv).	Necessary
iv.	The solution must store and reference Instance configuration data in the most efficient approach available. (see Instance Management chapter vii.)	Necessary
v.	Instance configuration must contain a target directory which should relate back to where the Application located.	Necessary
vi.	The target directory should be relative to the Application Environment dictating where inside the environment to access.	Necessary
vii.	Target directory is a non-mandatory configuration constraint	Necessary
viii.	The solution should support Instance modulation in the respect that id or other mandatory or even non mandatory variables can be passed from the Instance configuration.	Necessary

	Instance modulation variables should cater to any convention type, with moduleid being set as default.	
ix.	The solution should also allow for any manner of flags and configuration variables to be passed to the selected host entry based on configuration.	Necessary
x.	Instance modulation is a non-mandatory configuration constraint	
xi.	Instance configuration data should be cached and stored in memory rather than constant connections to the database for Instance identification.	Necessary
xii.	The solution must store SSL certificates relating to each host reference. The SSL references should allow for a many hosts to one SSL certificate relationship.	Necessary
xiii.	The solution should pass Instance data to the Application Environment for it to be used with Application Environment's inter-process communication (see chapter iii).	Necessary
xiv.	The solution should store authentication data and offer redirects to unauthorised users. Note: The Application will be the responsible for escalating the user authentication	Necessary
xv.	An API reference will need to be created documenting the use authentication process and procedures.	Necessary
xvi.	Documentation will need to be written explaining how to implement and configure Host Management.	Necessary

Application environment

Summary

The Application Environment is what is served by a web server to provide the user experience. In the case of PAL it needs to be a solution that can interpret all known scripting types or modularise to cater to any scripting types in the future. Case in point: PAL's key focus is to offer an agnostic solution to any pre-existing Web Application. The Application should be installed as normal and that Application, regardless of where it came from, should operate as a SaaS offering, including the ability to sell the offering.

The web server solution that needs to be selected in order to drive the Application Environment should also be an industry reliant solution. If problems take place within the source code, for example, an exploit of some type, the solution provider responds and has a strong history and track record of handling security issues. This also applies with general bugs. In the case of the web server, this will be the key public access point for customers and will be responsible for handling the most complex part of PAL. Therefore, it needs to have an equally strong solution supporting it.

Considerations

The key consideration is security. The second would be performance. After those two, other key considerations include:

- Compliance with web standards;
- Backward compatibility and legacy services;
- Support for multiple Application Environment configurations;
- Support for most common scripting languages and/or the capability of supporting them later;
- The ability to implement extended features such as Host Management;
- The ability to communicate and work with other associated sections or services (IPC); and
- Scale effectively within GNU/Linux / UNIX environment.

Multiple Application Environment Support

The system will need to support the ability to initiate different Application Environments based on different configurations. Each configuration will relate mainly to web server settings such as, scripting languages, compression types and transport types. It will also be used to delegate the location of the Application and other Application related settings.

Further, in respect to Instance based data, the solution should have the ability to store abstracted data for many Applications or just a single Application depending upon PALs use (see File System Abstraction chapter vii).

Inter-process communication (IPC)

As the requirement of working with other associated sections or services is needed a standardised mechanism for inter-process communication, such as message queues, is

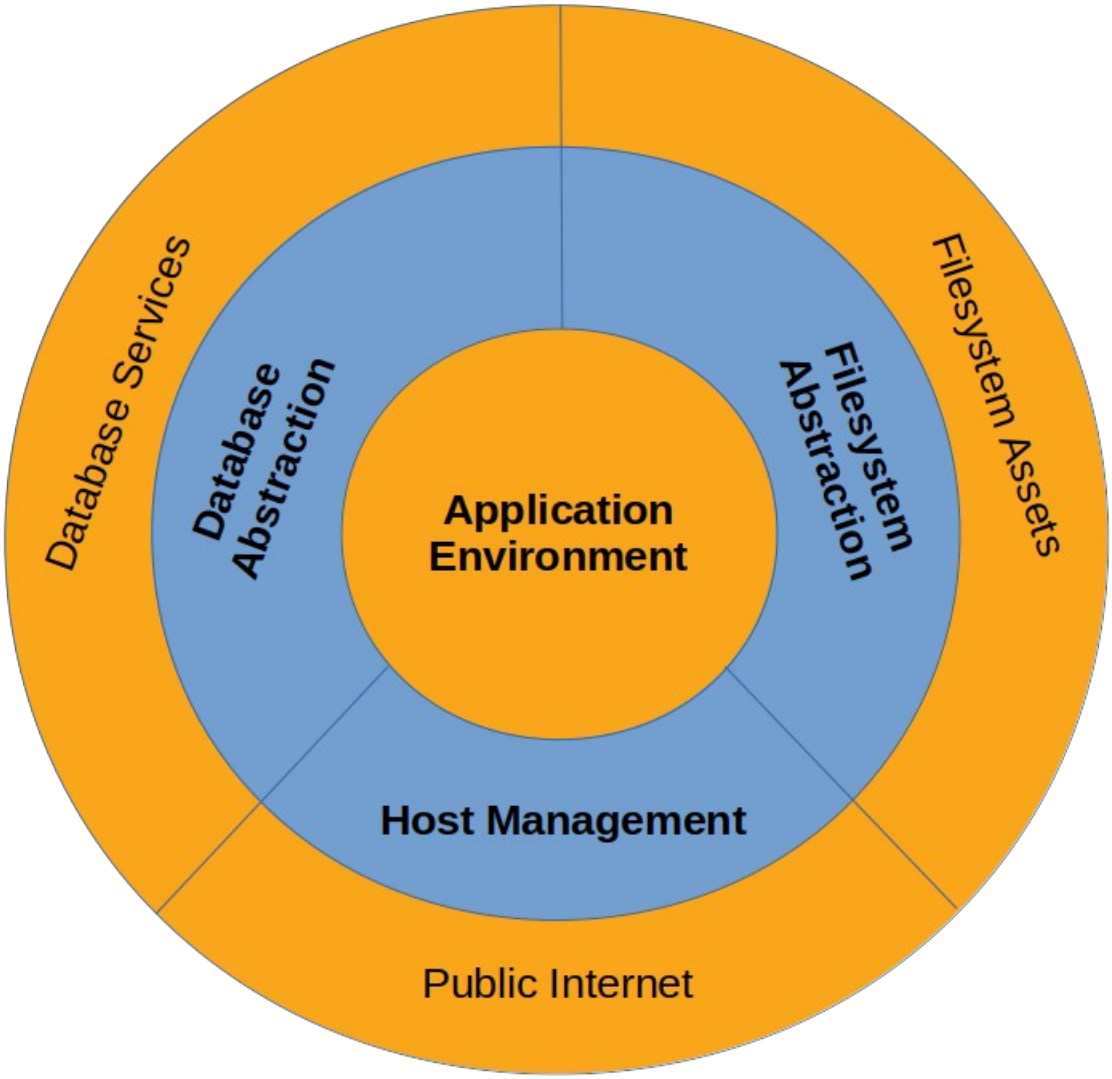
perhaps one method or approach that could be employed. (see https://en.wikipedia.org/wiki/Inter-process_communication https://en.wikipedia.org/wiki/Message_queue). Other methods could be used within IPC or even other parts of the kernel's ABI (see https://en.wikipedia.org/wiki/Application_binary_interface) to achieve the same cross process functionality. However, message queues seem the most feasible approach at this point.

Also, this part of the solution will need to act as separate sub-system or service that will sit outside of the Application Environment itself, and would act as an extension of the web server or even be a separate daemon provided it can easily interact with multiple Application Environments' running at the same time.

The key requirement irrespective of its implementation is that when the Application Environment executes an Application the associated sections and services, such as Database Abstraction, will need to know the Application Environment identifier along with the Instance, Application, module, access permission and so forth, or whatever is needed to use for that Application Environment to operate with. Further to this requirement the Application Environment will need to track each process and process id, and store and serve that data back to each the abstracted section or service when requested.

Graphical representation

Please see the following diagram representing the Application Environment ecosystem.



Functional specification

Please see the following articles.

#	Description	Class
i.	The solution must be industry reliant and must function in a UNIX or UNIX-like ecosystem. The preferred environment initially is GNU/Linux.	Necessary
ii.	a) The solution can be a pre-existing application such as Apache or NGINX provided it has the capability to modularise; b) The solution can be a pre-existing open source solution which is forked and extended for the purposes of this project; or	Necessary

	c) The solution is newly built.	
iii.	The solution should offer the most efficient multi-process environment and should create the Application Environment as a virtual machine, runtime environment, or container. For example see Apache MPM worker and MPM Prefork (http://httpd.apache.org/docs/current/mod/worker.html and https://httpd.apache.org/docs/current/mod/prefork.html).	Necessary
iv.	The solution must provide support for multiple Application Environment configurations meaning that the system should be able to support more than one Application if needed.	Necessary
v.	The solution must able to support most known language types. If it is unable to support a certain language type, the solution must have provisions that allow for further language types to be added, such as modulation.	Necessary
vi.	The solution should allow each scripting language to be enabled or disabled respectively per Application Environment configuration.	Necessary
vii.	The solution must serve web page data using known web server standards and compliance such as HTTP/2. The solution must also be legacy friendly and be forward capable of supporting HTTP/3 when necessary.	Necessary
viii.	The solution must have the ability to access user data across using operating system group based user management (GIDs)	Necessary
ix.	The solution needs to support most known transport type optimisations.	Necessary
x.	The solution needs to cater to the latest TLS/SSL support but also support older SSL certificate types and transport standards.	Necessary
xi.	The solution must support the latest in object and image compression formats such as JPEG2000 and WebP, GZip.	Necessary
xii.	The solution must be secure in the sense that if vulnerable code is found the response on patching is timely. Further, if it is a newly built solution then it must be built using the most secure practices available.	Necessary
xiii.	The solution requires a standalone service or module to act a messaging service for other associated systems, such as Database and Filesystem Abstraction (see IPC) . If this is not achievable a proposed set of conventions and/or references may need to be documented and created to allow for the configuration of the Application to connect effectively to such systems, for example Database Abstraction application level configurations.	Necessary

Database abstraction

Summary

Being the epicenter solution for data storage, databases are perhaps one of the most crucial parts of SaaS. Databases, meaning Database Abstraction, must keep security and then performance as its two most important roles.

Ideally referencing per Instance data can be done in many different ways. There are limitless conventions that can be used to categorise and handle SaaS data, also known as big data. PAL aims to work around four key principles and methods with the scope to invite future variations or completely new methods if deemed feasible. Furthermore, two of these methods are more or less combined as they lend from each other. The first method discussed is the simplest and most efficient of the four, however, it comes with its own limitations.

PAL's Database Abstraction will come in the form of a proxy solution that will be designed to accept incoming requests from the Application using Host Management instance ids and/or instance module ids to restructure messages to the database. Stored procedures should be evoked whenever possible to assist with improving access security or efficiency. Databases should also be designed in the most and secure approach with the correct use of mandatory constraints and most efficient and best thought-out indexing strategy.

Enterprise database technologies would undoubtedly have advanced methods which could be used to survive these overheads with proven track records. However, PAL comes from the standpoint of relying on free and open solutions such as PostgreSQL and MySQL. PAL attempts to utilise as many of the default security features of these solutions with the capability to take on more offerings including Enterprise level offerings if and when available.

Data storage methods and strategies

1. Separate databases per Instance

Each new Instance is allocated its own database, no lower level abstraction done, except for a few additional features such as flagging *deleted* or *archived* entries and time stamping data creation and modifications.

Databases are divided by using the instance id, as it is just appended to the end of the database name.

For example, *cars_db* becomes *cars_db_1* and so on.

This can also be done with the user account responsible for accessing the database.

For example, *cars_db_user* can be *cars_db_user_1* and so on.

Note: Although this may be considered as a SaaS component by some, by others it is not because it side-steps the multi-tenant rule (see chapter i).

2. Composite instance id and id is Instance data primary key

This method uses one database per Application rather than per Instance, creating a multi-tenant database solution.

Field	Details
instanceid	INDEX
id	PRIMARY KEY, AUTO_INCREMENT *
name	VARCHAR

*** Composite key indexing**

The use of composite key indexes (see https://en.wikipedia.org/wiki/Compound_key) in this method become the primary key (see https://en.wikipedia.org/wiki/Primary_key). Also, the auto-numbered indexing remains intact from what the Application would traditionally expect from the database running in a single Instance environment provided that the primary key id auto-increments in the following way:

Instance id	Id
1	1
1	2
2	1
1	3
2	2
2	3

With the proper use of indexing this method creates an efficient and production ready solution, however, maybe limited by the type of database software that is used. In the case of MySQL, composite keys are supported in this way (See http://dev.mysql.com/doc/refman/5.5/en/example-auto-increment.html).

Application level example query

update cars set name = “Holden GTS” where id = 2;

Database abstraction translated example query

update cars set name = “Holden GTS” where instanceid = 1000 and id = 2;

3. Instance id used to separate Instance data

This approach is similar to the second approach, however, is in reverse, as the use of composite keys are not used and the primary key id auto-increments against other instance entries or database rows instead.

Field	Details
id	PRIMARY, AUTO_INCREMENT

instanceid	FORIEGN KEY
name	VARCHAR

The drawback to this method is that across Instances the ids become compounded. Therefore, as the SaaS solution grows and takes on new Instances those newly created Instances will have high numbers as its primary ids. From a security standpoint this can pose problems as it infers cross Instance discovery (https://en.wikipedia.org/wiki/Discoverability).

Id	Instance id
1	1
2	1
3	2
4	1
5	2
6	2

Application level example query

update cars set name = “Holden GTS” where id = 5000000;

Database abstraction translated example query

update cars set name = “Holden GTS” where instanceid = 1000 and id = 5000000;

4. Composite Instance id and secondary id used as Instance data primary key

This concept basically takes concept two and three and combines them together.

Field	Details
id	PRIMARY, AUTO_INCREMENT
instanceid	FORIEGN KEY
secondaryid	INDEX, AUTO_INCREMENT *
name	VARCHAR

*** Secondary id composite key indexing**

Secondaryid auto-numbering will need to relate against *instanceid* iterations for its numerical indexing. However, due to the limitation that various database solutions may not support a secondary auto-incremented column, this kind of functionality can be created using stored procedures (https://en.wikipedia.org/wiki/Stored_procedure).

The below representation shows how the *instanceid* and *secondaryid* would work as composite keys in order to abstract the primary key database references for the Application.

Id	Instance id	Secondary id
1	1	1
2	1	2
3	2	1
4	1	3
5	2	2
6	2	3

Application level example query

update cars set name = “Holden GTS” where id = 2;

Database abstraction translated example query

update cars set name = “Holden GTS” where instanceid = 1000 and secondaryid = 2;

Implementation details

Support for non-integer key types will also be a requirement. In the case that Applications which already use composite keys as their primary key and wish to run under PAL. PAL will need to accommodate the ability to override it’s existing composite support and have the Application layer simply handle it, if needed. The limitation with this, is that instance id management goes through the Application and opens the door to cross instance database exposure if handled incorrectly.

Which brings us to the key advantage of Database Abstraction. PAL should come from the perspective of dissecting every query statement and reconstructing them around the concept of containment of Instance data through Host Management constraints (see chapter ii). This restriction can be table or id based, and is determined by the host entry data, which is passed via Inter-process communication, and should include the following:

- Instance id
- Application Environment id (*if needed*)
- Instance modulation ids
- Authorisation and UID / GID access.

Furthermore, there can be varied ways Database Abstraction can be used. A combined approach is the most transparent and technically viable, having all four methods operate together. Although, it is acceptable to just use one of these approaches independently and scale for redundancy and performance.

Further, because each method can have different use cases this means that separating, staggering, or even cloning methods should be a functional requirement.

Performance considerations

Since stored procedures and other methods, such as caching or hash tabling (https://en.wikipedia.org/wiki/Hash_table) can be employed to help with the abstraction process. Any supplemented databasing concept which handles data by the means of

computational tasks rather than direct disk access operations cause overall performance cost and inefficiency, and this is where the art efficient databasing will come in to play.

The most effective form of database optimisation will always be the use of effective indexing (https://en.wikipedia.org/wiki/Database_index). Indexing will always be the preferred and recommended form implementation strategy. Therefore, offering different configuration designs or connection profiles, based on the database solution and optimisation strategy will be important.

Configuration constraints and profiles

Connection profiles will be the term used to describe the different method configurations relating to most the appropriate database solution for that method type. For example, MySQL may only use method two and three, whereas PostgreSQL may only use method three and four.

The following methods described below give PAL's Database Abstraction the flexibility to factor in as many configuration scenarios as possible. Thus, allowing little or no interaction at an Application layer and simplifying the database component for developers. Meaning that developers only have to focus on creating single Instance Applications with PAL, as Database Abstraction will take care of the multi-tenant component autonomously, and attempt to do so in the most efficient and effective way possible.

Combining methods

The simplest and most ideal scenario is as follows. Having all four solutions operate in tandem with one another and each method having basic *read* and *write* rules applied based on their purpose. For example, when an Instance writes to the database PAL can *write* to methods one, two, three or four (three and four being the same write) at the same time. However, only method one is used for *read* operations for that Instance. That is, method one handles of all the SELECT requests while all methods accept INSERT and UPDATE requests within the Instance. While outside the Instance, Instance Management or direct database interactions can have complete access to all data in a combined format.

Secondly, although method one is perhaps the fastest and most secure, it does not allow for cross-Instance statistics or cross-Instance communication. That is, only data per Instance is accessible with method one, while the other three methods allow for cross-Instance data consolidation and access.

Lastly, method four and method one can have method one's primary key match method four's secondary key and can be used to create a crosscheck to ensure synchronicity between all four layers of Database Abstraction.

Separating methods

It may be required in certain scenarios that instead of combining methods each method may need to be called by the Application separately so that even though a unified method may scale well there can be situations or even entire solutions that may want to do multiple SQL calls for each individual method independently.

Staggering combined methods

In other scenarios a staggered approach may even be considered where some data is retained in one method but left out from others. Therefore, a process will also need to be devised that will allow fields or tables to be whitelisted or blacklisted between methods.

Sibling and cloned methods

Since the system will cater to concept of connection profiles. Methods should be reusable and customisable, therefore, should also allow for consignment next to each other either in a combined, separated, or staggered combined approach. Another way to explain this is that one identical method could be used in different connection profiles and those connection profiles should work with each other depending upon configuration.

Convention translation between methods

Lastly, having the ability to have the same field with different names across methods should also be a functional requirement.

Asynchronous operations

Since PAL aims to abstract as many processes as possible and be as efficient as possible at the same time, creating a solution that closes the server connection after one method has completed its operation, meanwhile expecting subsequent methods to “catch up” in the background. Therefore, having asynchronous connection profiles should be a manually configurable option within each connection profile, with the forced condition of having one profile as the primary synchronous request.

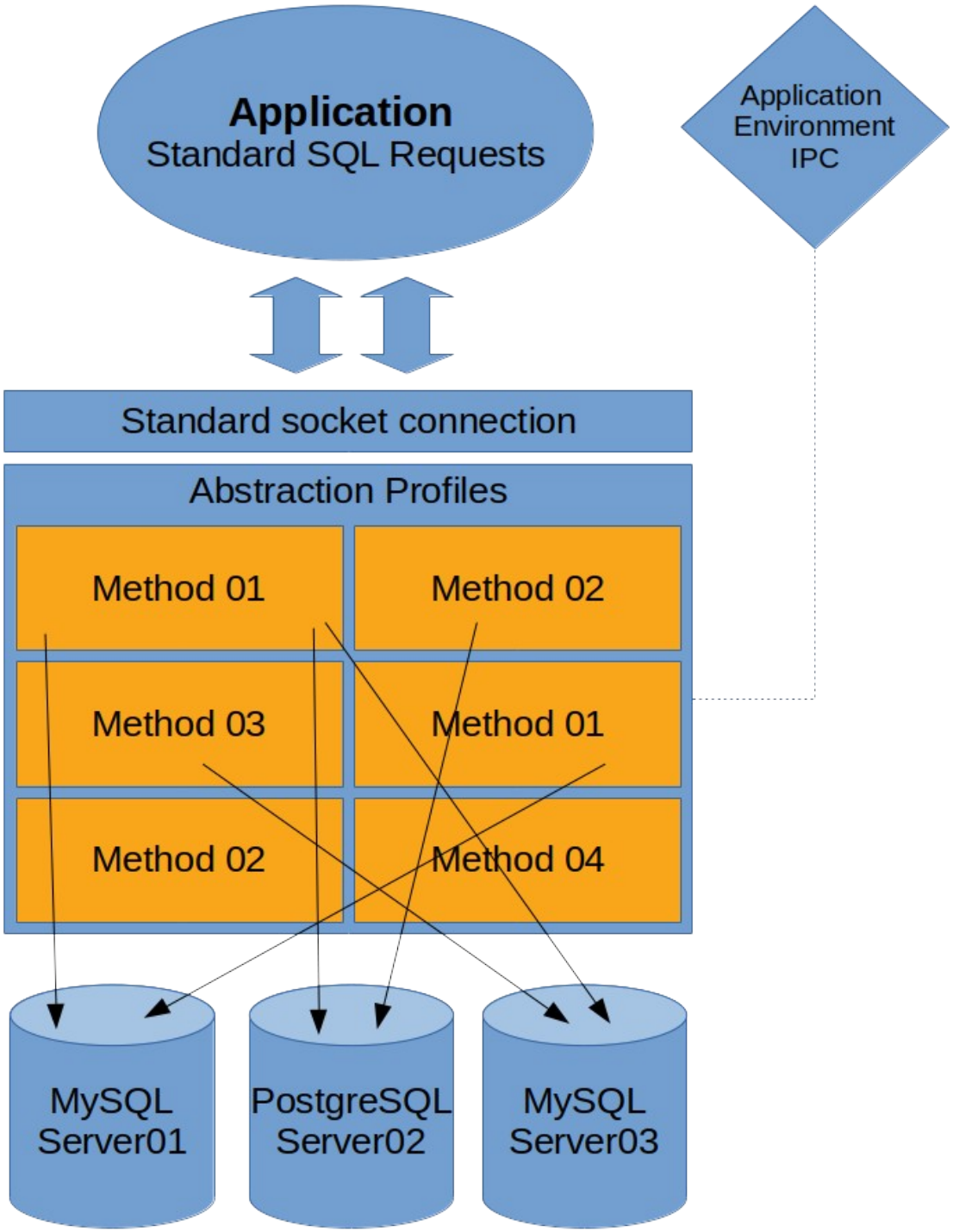
Extended abstraction features

Data retention is an important feature of any SaaS platform for various reasons, the most important being for recovery purposes. Extended abstraction features should include the bare minimum requirements:

1. Items or columns are to be flagged deleted with *bool* rather than using DELETE, enabling entries to be resurrected by an system administrator;
2. An archived (*bool*) flag is used to archive data which should be excluded from certain requests, useful when process intensive data needs to be accessed; and
3. Track creation date and last modified date with timestamps

Graphical representation

The following diagram represents Application level standard SQL requests are being sent and received.



Functional specification

Please see the following articles.

#	Description	Class
i.	The solution will be: a) Created as separate service or daemon; or b) Created as a module inside the nominated web service solution	
ii.	The solution must accept Instance identification data from Host Management before accepting a request or connection from the Application.	Necessary
iii.	The solution must interact with the Application Environment via IPC (see chapter iv).	Necessary
iv.	The solution must cater to and unify all methods and approaches to work independently with each other: a) Separate databases per Instance; b) Composite instance id and id is Instance data primary key; c) Instance id used to separate Instance data; and d) Composite Instance id and secondary id used as Instance data primary key	Necessary
v.	Abstraction profiles should be created meaning that abstraction methods are created with distinct configurations. Profiling should allow for multiple abstraction methods of the same or mixed type to be used in parallel with one another.	Necessary
vi.	The solution should support most open source database services with the initial offering being MySQL, PostgreSQL and MangoDB.	Necessary
vii.	The solution should use whatever optimisation methods which are offered by each Database software offering, eg Hash tabling, stored procedures, caching.	
viii.	The solution should support non-integer keys allowing string based ids and composite key data types.	Necessary
ix.	The solution should support convention translation between abstraction profiles	Necessary
x.	The solution should support whitelisting / blacklisting of column names within abstraction profiles	Necessary
xi.	The solution should allow for collaborative and partial cross method support. For example, database read requests are only done by one abstraction profile while writes are done to all abstraction profiles.	Necessary
xii.	The solution should use stored procedures or other methods wherever possible to security requests For example, a stored procedure can be used to grant group access to certain ids.	Necessary
xiii.	Create basic abstract data attribution for data preservation and tracking purposes for all column entries throughout all tables: a) Flag when items are deleted with a *bool* rather than using DELETE; b) Flag items for archive with a *bool;* c) Flag created timestamp; d) Flag modified timestamp on last column transaction; e) Flag IP address (optional).	Necessary

Filesystem abstraction

This chapter will deal with flat file storage both for the actual Application and for any customisations made to any particular Instance.

Creating an independent service that abstracts Filesystem access would offer various advantages such as permission separation, however, would require significant work when tracking instance data, directory permission data and process separation. Meanwhile, a simpler approach would be to make Filesystem Abstraction a part of the web service in the way of a module or extension.

Irrespective of the design, the main role of Filesystem Abstraction will be to handle all *read* and *write* changes made to or by the Application, or Application Environment. PAL aims to store the modified elements for each Instance separately while the Application itself is accessed as read only.

The following two approaches should be considered:

1. Instance data housed under one user account

The Application tree structure as follows:

/home/user/www/applicationY (location of the applicationY data)
/home/user/data/1/applicationY (location of instance id applicationY's data)
/home/user/data/2/applicationY
/home/user/data/3/applicationY

And so on …

2. Instance data housed under different user accounts

The Application tree structure as follows:

/home/applicationZ/www/ (location of the applicationZ data)
/home/instance1/dataZ (location of instance id applicationZ's data)
/home/instance2/dataZ
/home/instance3/dataZ

And so on …

The best approach

Ideally both approaches should be considered, however, the second approach allows for added security through the use of account user creation and group access (UID / GID) via the operating system. The second method also ensures file permissions are more resilient by attributing an extra layer of permission control via user credentials. The second method also means that other services can be used safely to access Instance specific data. For example, SSH and SFTP. This also means that Instance owners can retrieve or modify their data directly without having any direct access to the Application.

Regardless of how the data will be stored, whether it be approach one or two, PAL will need to mark certain files or directories with certain flags to ensure that Application updates do not cause any disruption or a negative impact to Instances.

Volatile, Constant, Sequential and Standard

During software updates, the behavior of Instance based data may require treatment by an integrator or administrator. Or in other scenarios, the Instance may inadvertently try to rewrite parts of itself that need to simply stay non abstract. Therefore, the following flags have been devised and can be applied on files or directory structures as needed:

Volatile - Files or directories marked as Volatile will mean that updates will require each file to be manually checked by the integrator, administrator, or customer.

In many cases a Volatile file will not need changing. An example of a Volatile file is a template file that may have variables from the controller removed or added and could cause an exception or error once the new version of the Application is loaded in to production.

Constant - If a file is marked Constant, it means it is an untouchable file or directory by the integrator, customer or Application. For example, Controller classes are stored in the */controller* directory. Therefore, everything under */controller* can be marked as Constant and cannot be overridden at any level.

Sequential - Similar to Volatile in the sense that if the file or directory is marked sequential and the timestamp on the Application version is newer, the newest version of the two are displayed, ie in case the Application version update, and the Instance specific version is only made available after a simple touch command on the file or an after the file is updated.

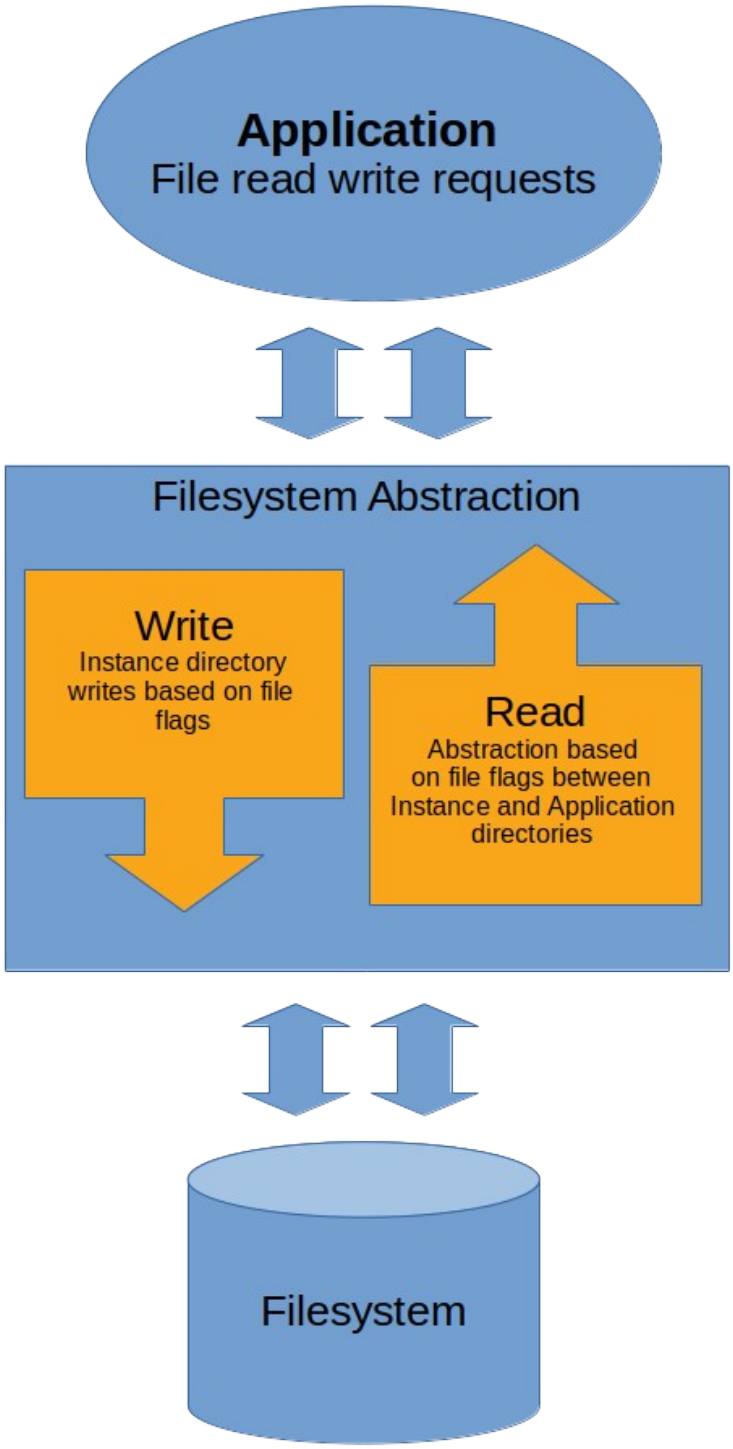
Standard – Lastly, if the file is flagged as Standard this means that the simple abstraction rule applies regardless of version or update history of the file and is simply abstracted irrespective. The draw back with this flag is that unexpected behavior such as Application level errors or exceptions could be thrown causing bugs or glitches to reside within particular Instances.

File permission mapping

The ability to manage file permissions based on the operating systems' access control (https://en.wikipedia.org/wiki/Access-control_list) will be an important feature of Filesystem Abstraction. An example of this is in the case of Instance data, where the Application itself may have writable components that need to be shielded from integrator. In such a situation, PAL should also allow for mapped permissions to work inline with the operating systems' access control and offer a custom permission structures to be mapped out and utilised when required.

Graphical representation

Please see the following diagram.



Functional specification

Please see the following articles.

#	Description	Class
i.	The solution will be: a) Created as a module inside the nominated web service solution which delegates file system requests. b) Created as separate service which interacts with the Application Environment via IPC, or another consigned method, which will intercept and abstract Filesystem requests based on process origin; or	Necessary
ii.	The above referenced methods of abstraction should be catered to: a) Instance data housed under one user account; or b) Instance data housed under different user accounts.	Necessary
iii.	Application source data should be stored as read only and writes should only occur within the Instance data location.	Necessary
iv.	The solution must cater to the above described file system control flags: Volatile, Constant, Sequential and Standard.	Necessary
v.	The solution must reference relative path data received by the Application Environment and should abstract the Instance path over the top of the application path. The behavior of this should be controlled by the above described flags	Necessary
vi.	The solution should be able to reference absolute data in the same fashion as point iii but should be a global enabled feature with the ability to disable the feature per Instance or vice-versa.	Desirable
vii.	Unspecified Application Environment flat file data that may step outside the currently working directory such as /tmp file data for PHP will also need to be stored within the Instance data inside a separate location.	Necessary

Instance management

As PAL requires a service to administer Instances and maintain the server health and performance, a web-based portal which will be accessed by Host Management will need to be created. Host Management will not only use a special flag in its URL entry but this URL should remain secret from robots and web crawlers.

The Instance Management Portal should maintain the following features:

System based features

- General configuration for the following sections:
  - Application Environment;
  - Host Management;
  - Database Abstraction; and
  - Filesystem Abstraction.
- Per Instance Application level task scheduling.
- Instance management and migration tools.
- Database administration and schema modifications.
- Basic backup and recovery scripts.

Instance management features

- General Instance configuration.
- Global, grouped or per Instance based version control including the ability to dynamically roll back or forward Instances.
- Instance based resource consumption and quota management.
- Instance and Instance module activation (REST / JSON).
- Basic license management features.

Functional specification

Please see the following articles:

#	Description	Class
i.	The solution needs to be written in an industry accepted web framework, such as Symfony or Flask.	Necessary
ii.	The solution should use industry accepted client side framework and/or libraries to handle the user experience and interface, such as Vue, AngularJS, Bootstrap, PureCSS and so on.	Necessary
iii.	The solution needs to have an authentication regiment which can allow for user-based access restrictions. Accounts with granted permissions should allow for reseller or partner accounts and logins to be created.	Necessary
iv.	The solution should be skinnable, meaning that aesthetic components of the solution should be easily customisable as to allow the provider the ability to create a look-and-feel representative of their brand. See https://en.wikipedia.org/wiki/Skin_(computing)	Desirable
v.	The solution should be developed with the functionality of Progressive Web	Desirable

	Application (PWA).	
vi.	The solution should be SSL encrypted by default with non-SSL connections redirecting to SSL.	Necessary
vii.	Two factor authentication (2FA) should be an extension or plugin.	Desirable
viii.	The solution should be built to withstand brute force attacks by blocking failed attempts.	Necessary
ix.	The solution should allow for any APIs or third-party connectors to use separate delegated URLs for service access to eliminate network discovery.	Necessary
x.	The solution needs to encapsulate the following features.	Necessary
xi.	General configuration for Host Management.	Necessary
xii.	General configuration for Database Abstraction.	Necessary
xiii.	General configuration for Filesystem Abstraction.	Necessary
xiv.	Tasks scheduling for any application-based tasks that need to take place. Task scheduling should come with the feature to allow for the task to be executed as a per Instance.	Necessary
xv.	Instance management and migration tools.	Necessary
xvi.	Database administration and schema modifications.	Necessary
xvii.	Basic backup and recovery scripts.	Necessary
xviii.	General Instance configuration.	Necessary
xix.	Global, grouped or per Instance based version control. Including the ability to dynamically roll back or forward Instances.	Necessary
xx.	Per Instance based resource consumption and quota management.	Necessary
xxi.	Per Instance and Instance module activation (REST / JSON).	Necessary
xxii.	Basic license management features.	Desirable

Customer licensing and resources

Ideally any business model which surrounds the Application should be of little consequence to PAL. This means the broadness approach must be invoked at this point.

It is simply better to offer a uniform approach to how SaaS can be packaged from a barebones technical standpoint. Therefore, makes resource and resource management the key areas where PAL aims to address when it comes to customer licensing requirements.

Resource allocation methods

Per module activation

Since modules can be used (or not used) for each Instance this should be a deciding factor on how resource management can be divided. For example, the disk quota size of a shopping cart module may be bigger than one for a blog module and therefore should allow for different per module per resource allocations.

Database consumption

Database consumption can be gauged two ways:

1. The number of primary listings (For example, a shopping cart *products* table would be the primary listing table, therefore, you can sell 1000 product shopping cart); or
2. The database blocksize per Instance / module offering.

Data quota management

Ideally, each Instance directory can have its blocksize read from the Filesystem and used to measure quota size. Since, modules can store data in neighboring directories, this can be separated per module if needed as well.

It is important to note that the system will need to hold three types of quota information. The first will be that of the data consumed by the current active version of the Application. The second will be the quota of entire Application itself and lastly the quota of the entire Instance. Since, in many cases the current active version is what is sold to the customer. That quota is what will be used from a productisation standpoint.

Bandwidth quota consumption

The web server's logs usually hold this activity and recording these data increments can be made possible. Although, since it requires passive data handling to record bandwidth. This feature may come at a performance cost and all methods leading to its implementation should have all possible optimisations taken into consideration.

Instance signup and demos

SaaS doctrine makes way to various new norms found in the market today. One of these said norms of SaaS is the concept of *X days for free* of an Application as a demo or trail account. In this case the customer is expected to provide a credit card number during the initial sign up and when *X days* have expired begin to charge the customer. The alternative to this is allow the customer to sign up with limited sign up info and allow the Instance to run for *X days* and then request payment leading up to the date of expiration.

Either way, PAL will not aim to provide answers but instead give a framework to allow for as many combinations as possible. The signup process should come in a way of a REST web service which has been given authorization to create accounts. That service should be part of the administration and monitoring aspect of PAL's Instance Management segment. As for the actual billing regiment, when an account is created, suspended and so on, will be up to the actual SaaS provider.

Application version control and application staging

An important feature for Instance Management will be the ability to handle different versions of the Application. Since there are many ways this could work the following scenarios that should be catered should mainly focus around Application data and Instance specific data.

Ideally things such as symbolic linking can be used to simplify this approach. However, referencing directories within the Application Environment should be the most critical feature, as to allow for an individual or even a group of Instances to be rolled back or forward to previous or newer versions. Therefore, storing directory references per Instance within the Application Environment will be important. If symbolic links are used this is acceptable provided it holds to a per Instance relationship and not just a static reference.

A key issue will be when it comes to quota management after committing updates. Since copying data from one directory to another directory will increase quota size this poses as a disadvantage and cost of Instance consumption. Furthermore, the critical factor of checking any Filesystem Abstraction flags such as volatile or constant will need to be addressed to at this point. These checks should be in line with the ability to manually roll forward Instances as each series of flags for that Instance have been accepted by the Administrator or Integrator.

In the case of Application staging, PAL should allow for seamless staging environments and to cater to all development approaches. Ideally staging can happen on separate servers but issuing updates to production safely will be the most important process moving forward.

Either way, the Application Environment and Host Management sections should be configurable to swap out Application directories easily without affecting Instance data and directories which contain duplicate or reused data should not be included as the quota which is sold back the user.

Instance data should be structured in a way where separate directories can also be used at this point of upgrade and the retrospective directories are simply not included unless rolled back. However, this could also work in the opposite by updating the code to the latest version and then selectively upgrading Instances manually. If, for example, the system requires an update and there are Instance data changes from version X to version Y. The following directory structure before the update would be ideal.

/home/instance1/application-data-version-1.0.1/
/home/instance1/application-data-version-1.0.2/ (Selected)
/home/instance1/application-data-version-1.0.3/ (which at this point is a copy of 1.0.2)

/home/application/www/application-version-1.0.1/
/home/application/www/application-version-1.0.2/ (Current)
/home/application/www/application-version-1.0.3/ (Latest)

The system checks the Filesystem Abstraction flags for application-*data-version-1.0.*
Once satisfied the operator / administrator is satisfied then Instance is rolled over to the newer version 1.0.3

It is important to note that this rolling update process should be a per Instance requirement as to allow for individual roll backs or forwards to a previous or future version.

The directory mappings that are be handled directly by the Application Environment will allow for this to be easily achieved, and that's why if mapping is done with a symbolic link, that symbolic link relationship is a per Instance symbolic link and not a universal definition.

Database administration and schema updates

Instance Management should be built with a simple schema update tool that will work with PAL's Database Abstraction. Third party schema modification software can be used for adding and removing individual columns. Provided that the initial schemas are created by Database Abstraction and then imported into the software, changed, and then updated.

The rule being that each abstraction method will need its own schema and all updates will need to be modeled on the definitions laid out by those individual abstraction methods, and therefore may require multiple schema updates. However, the ideal approach would be to allow Database Abstraction to translate any table modifications to each consigned method by having a tool that translates table modification requests and then updates each method automatically.

Furthermore, backups or snapshots of the previous version of the database(s) should be issued before any updates are made. Those backups should be either timestamped or version stamped and correlated to their respective versions. However, the latest database should always take preference to avoid data inconsistencies and rolling the database per Instance backwards is only done manually if absolutely necessary.

System maintenance

System maintenance will ideally be what is prescribed in Instance Management such as the ability to version control Instances, group manage Filesystem Abstraction actions, and it should also have the ability to plug in other solutions. For example, database storage services (MySQL, MangoDB and so on).

Integration and third-party services

Such services should be contrived in the Application rather than within PAL. For example, a web service that ties XML based booking availability to an Application should be handled and written within the Application alone and any service which is hosted by PAL should simply work without any modifications.

Furthermore, things like GDPR compliance and the extraction of data should be made easier with the ability to allow for any open SQL requests to be handled via the Application as well. For example, an open input area that solicits SQL SELECT statements and variations of SELECT can be made available to the user of that Instance's safely. The user should simply be able to export all data or even import all data, depending upon what is prescribed by the Application and Application developer without any security concern.

Another consigned methodology for integration look-and-feel should also be the possibility to create an Application that does flat file saves to system templates and then allow the Application, again, to handle the saving, overriding and deletion of those files. Meaning that customers, not just integrators can make look-and-feel modifications to the Application when needed.

Consumer data

The topic which puts the most controversial spin on SaaS, or even on Cloud Computing in general, is usually focused around customer data protections. These topics usually include data ownership and sovereignty and not just technical considerations, such as security and efficiency. From a SaaS providers perspective, the provider should already cater to the philosophy and standpoint that the Instance owner should have access to their data and that the data should be provided in a responsible format. The semantics of this go deeper when talking about security and having to offer data back to the customer. Ideally flat file representations in CSV format with neutral column definitions should be acceptable on many situations.

The problem with all this is when the time comes for a customer to switch providers and data entry is required instead of a feasible way to import the data automatically.

Inherently offering the most open export process per Instance will always be the best thing a provider could ever offer, and with PAL's Database Abstraction, providers should theoretically be able to accommodate direct SQL imports and exports inside a secured administration environment and pose very little security risk to the provider.

It also means that Database Abstraction can limit access to certain tables and protect parts of the Applications' anatomy which could be considered propriety elements or security hazards. Ideally the purpose of PAL will be to maintain the most open standpoint as possible. It will be up the individual provider and Application feature-set on how much the Instance data the customer can access.

Final thoughts

Project Abstraction Layer attempts to fill a gap in the market by creating a tool which has various use cases in both a commercial and noncommercial setting. The solution is intended to be a tool that can convert any prebuilt Web Application into a full cycle SaaS deployment solution.

This gap has remained relatively open since the inception of Software as a Service and has attempted to be catered to by various out-the-box solutions over the years. Although many tools have been developed to help simplify and try to cater to this process there is no one unified solution that will a take piece of already contrived software and convert it into SaaS. Furthermore, even if there is, there still is not one unified solution that can help productise and manage licensing for a SaaS service.

If you consider that Platform as a Service (PaaS) falls in line with the traditional Dedicated Server provider role, Infrastructure as a Service (IaaS) falls in line with the concept of selling Server Rack space. Then Project Abstraction Layers' philosophy sees that Software as a Service solutions should be treated similar to the way a traditional shared hosting provider works with reseller accounts and user accounts dividing permissions, resources and services.

License

Contributions

Since the project is just a functional specification at this stage any and all contributions, feedback, enquiries, clarifications and so on is welcome at this point.

Please see the below contact details.

Email: jamestbiv@gmail.com
https://abstractionlayer.jamesbiv.tech/