## Customer Relationship Self-Selection and -Maturation Process

A customer goes through a number of stages, and can stay at any particular stage for an indefinite period of time. In general terms, these stages are:

* Session
* Verified
* Autonomous
* Enterprise

When a potential customer first visits the site, they are able to anonymously use the site to build and operate an application. This initial usage could be considered a *trial* usage, but instead of classifying the customer as a trial customer, it is better to classify the customer by how the site operates on their behalf. Specifically, as an unknown user, the system only allows access to the application being developed and/or operated from the user’s *session*. This term has a number of technical meanings, but we use the term to describe a web session that is defined by some combination of the device used to access the site and/or the IP address from which the access occurs.

There are a number of reasons for this model. First, the usage of the site needs to be as *frictionless* as possible. That means that the user should not have to provide any information that is unnecessary, redundant, or confidential, unless there is a compelling reason to do so. This is, of course, at odds with the basic tenets of customer relationship management, in which we – as the site owner – wish to quickly and clearly establish the identity of each user, and to collect the maximum amount of information possible, moving the user rapidly to a purchasing decision, and providing us with the maximum amount of information and leverage to accomplish that goal.

Session: Without providing any information, a developer can begin developing an application and experimenting with the platform. The session lasts for at least 30 days, and allows the developer to move between work and home, for example.

In the model we are proposing, the user self-selects the points at which identifying information is provided, based on a *quid pro quo* model. For example, it is our stated policy that for security reasons, an application cannot be publicly accessible without identity verification. Specifically, to comply with various mandates (imaginary or otherwise), we will need to collect a full name, an email address, a mobile phone number (for text messages), and a verifiable credit card number in order to allow an application to be accessed from arbitrary devices and locations other than those associated with the *session*.

As part of collecting this information, we seek to establish in the mind of the potential customer an image of our business and how it works. Specifically, while we will collect an email address, we will explicitly promise to never sell it, to use it only when necessary to contact the user about their account, and so on.

Similarly, with respect to the phone number, it is necessary for “second factor authentication”. Specifically, we wish to avoid requiring the user to create and remember yet another password in order to interact with us, and the phone can serve as a secondary means of communication and authentication in addition to the email address. Again, we will explicitly promise to never sell it, and to use it only when necessary based on the preferences that the user has provided to us.

For developing an application that exceeds the *session* scope – for example to allow multiple developers to access the application, or to ensure that the application is stored persistently, or to select a license other than the default open source license – the user must provide a name, email address, and mobile phone number.

Verified: In return for providing a name, an email address, and a mobile number, the developer can select a project license, open the project to other developers, and the project is saved.

Considering the risks of credit card fraud, and the expected desire of most of our “customers” to avoid paying for anything, getting a user to provide a credit card number will be challenging. Again, the request will be presented as a security measure, in terms of authenticating the user by correlating them with an account (whose bank one would expect to have authenticated the person at a fairly thorough level.) Our goal here is to obtain a means to bill the customer for small amounts in the future, in order to minimize the friction of subsequent purchases. As with other information, we will promise not to sell this information, and far more importantly, we will promise to never charge the customer without their explicit **pre**-approval. In other words, a customer can give us their credit card number, and we will never charge it, unless and until they explicitly approve a charge in advance.

In order to operate an application, which is to say in order to make a running application publicly accessible, full verification including a credit card number will be required. Additionally, a click-through will be used to provide the user with an understanding of the licensing, pricing of various options, and the quality of service that corresponds with each option or lack thereof.

A running application consumes resources, including compute, storage, and network. Resource utilization and aggregate usage is carefully tracked, and carefully controlled; it is possible to limit resource utilization on a resource-by-resource basis. A certain resource level is provided to verified users for no fee. It is acceptable that a large percentage (on the order of 99%) of users are *free-loaders* (in the non-pejorative sense of the term), because their usage of the site uses trivial amounts of resources, and they represent actual value in platform adoption and the resulting *network effect*, far more than any revenue that they could possibly represent. Note that the revenue that they could possibly represent will often be uncorrelated with the value that they derive from the site, but the point that we will focus on is *not* the value that they derive from the site, but rather the amount that they would be willing to spend to achieve that value; only in the unlikely case in which that small amount of spend exceeds the resulting perceived value will we lose these customers, which is arguably a desired result.

Paying customers are self-selecting, because they desire more capabilities or resources than are available for free. Specifically, when a user exceeds the resource limits that are available for free, they are notified (using email and/or text message, as previously specified) and given the *option* to purchase the additional necessary capacity. Failure to do so will potentially degrade the application (in as graceful and non-intrusive manner as possible), as one would logically desire if one were a customer.

Because all of the interaction thus far is automated and self-service, these now-paying customers are considered to be *autonomous*. Their cost of service is nominal by design, and the revenue is largely intended to cover the variable cost of hosting the site, with some additional margin. The customer always has the choice to take their application and go somewhere else, so a pricing model that makes leaving look like the expensive and painful choice is going to be key to long-term customer and revenue growth.

Autonomous: In return for providing a credit card, a developer can have their application hosted at no charge, up to a certain capacity and service level.

Lastly, enterprise accounts are the only type of customer accounts in which there is a human relationship, whether sales, support, or otherwise. The entry price point for an enterprise relationship is dramatically higher, perhaps on the order of US$1,000,000 per year. To avoid pushing companies toward enterprise relationships, the number of *autonomous* capabilities will need to grow over time, including for things like:

* Support for purchase orders and invoices in lieu of credit cards;
* Support for complex identity and security configurations that support arbitrary numbers of groups and users, in a manner compatible with business IT organizations;
* Support for interoperability with existing company systems via VPN style technologies, etc.

Enterprise: For organizations making extensive paid use of the platform, and desire a human relationship, or have requirements that are unmet by the autonomous platform

Over time, as more and more applications become available on the platform, there will also emerge a significant differentiation between *developer* users – those building applications – and *operating* users – those simply running applications that were already developed and made available. As and when this occurs, the workflow must be streamlined for each of these two user types in order to minimize the friction of utilizing the site.

In all cases, a fundamental goal of the platform is to provide an incredibly rich set of capabilities, and the means to effectively master and consume those capabilities, without requiring any involvement by employees of the company. It is an explicit business goal to minimize the percentage of customers that are forced to move to the Enterprise stage, and it is expected that continuous platform evolution will be required to augment the Autonomous capabilities accordingly.

## Licensing as both an Attractant and as a Barrier to Entry

As with the pricing model, in which we accept as self-evident that developers will not adopt anything that has an up-front cost, with respect to licensing we accept as self-evident that developers will not choose to adopt closed technologies. The language and the platform must be open source.

Not all open source licenses are created equally. For sake of context, here are the relevant categories:

* BSD, MIT, Apache, etc. – Use of the code is fairly open-ended and restricted only by requirement for inclusion of the license or copyright notice. Generally referred to as a “BSD-style license.”
* LGPL, GPL with Classpath Exception – A license that is roughly as liberal as the BSD-style licenses in terms of using (linking to) the licensed code, but any changes to the code itself that are distributed must be published or provided back under the same license.
* GPL – The prototypical copy-left viral license, that requires any dependent or otherwise linked code that is distributed to also be provided under the GPL license.
* Affero GPL – An extension to the GPL license that triggers the distribution cause when the code is hosted (i.e. even if it is not distributed); in other words, usage of Affero GPL licensed code in a hosted environment requires any dependent or otherwise linked code that is hosted to be likewise licensed under the Affero GPL, and the source code thereto to be made available.

The Affero GPL license was created to address a loophole of the GPL that allowed major technology hosting companies, including Google and Amazon, to take GPL code, modify it for their own benefit, and use it extensively, all without having to provide back any of their improvements, and all without being affected by the viral nature of the GPL license (i.e. they did not have to publish any of their own linked work under the GPL license) – all because those organizations did not *distribute* any of that code to other users. In order to close this loophole, the Affero GPL license triggers the GPL distribution clause even when an organization is only *hosting* the code.

The company will employ a combination of open source licenses, and retain a few key components internally, to allow the vast majority of the platform to be published “in open source”, yet without allowing a potential competitor to pick up the entirety of the work to compete directly with the company. This is a delicate balancing act, since we do want companies to be free to pick up the technology and use it to their advantage, and anything that we do to discourage that will undermine our broad adoption strategy.

Importantly, in order to provide the code under various licenses, we will need to have copyright over the code, or have been granted the right to use and relicense the code in any of the manners that we are describing herein. This would allow us, for example, to publish some of the necessary code under the Affero GPL license, while allowing us to host and distribute that same code (or finished products built from it) under any license that we choose (including commercial licenses), and without any obligation.

As a rule of thumb, general technology components, such as the language, the language runtime, and the core libraries, will be licensed under a BSD-style license. This encourages adoption, allows experimentation, and creates a generally level playing field (which in turn encourages adoption).

There will be specific, advanced features of the language runtime and the hosting platform that will be retained as proprietary code. While these features will be transparently available to users hosting on our site and for those obtaining a commercial license, they will not be made available as part of the open source code base. However, these features will be carefully designed so that they do not change the public APIs for the runtime (and the APIs will have been designed with the potential for the existence of these features in mind). Application developers will likely be unaware of the features’ existence, other than as is evidenced by the characteristics of the site and/or their running applications. One specific example is related to resource usage measurement and management, a differentiating capability that will allow us to dramatically increase the density of hosting (the number of customers and applications that can be hosted per server, per VM, etc.), which is a critical enabler of our business plan, since the vast majority of our customers will never pay for the services that they consume, and those that do pay will likely do so because of the low cost, which in terms provides a compelling cost-to-benefit ratio relative to competing alternatives.

The copyright for the core intellectual property for the language, the language runtime, the various standard libraries, and the site-hosting platform (collectively, “Core IP”) will be held in its entirety by the business, and by no one else. This will allow the business to determine how and when to license out the Core IP, including the selective use of both open and proprietary licenses. A series of open source projects will be organized around the language runtime and platform, which is straight-forward because the vast majority of the code will be available via one or more open source licenses. In order to accept contributions from outside of the company – including new features, improvements, and bug fixes – a contributor’s agreement will be used that provides the company with a grant of copyright (or a legal mechanism that accomplishes the same effect) for the contributions. To ensure the “cleanliness” of the IP, the dependencies of the Core IP will be minimized, and only carefully vetted licenses (MIT, BSD, Apache, etc.) will be acceptable.

Source code (etc.) is organized into *projects*. Each project has meta-information that is managed on its behalf, such as who the creator and/or owner of the project is, who is able to access and/or modify the project, what the history (sequence of changes, etc.) of the project is, what dependencies the project has, what license or licenses the project is available under, and so on. Customers can create projects in order to try out the new language and platform, to experiment, to build reusable components and libraries, or to build applications. The first few projects will be created by the company to publish and manage the public portions of the Core IP code, but as customers begin to investigate the platform, the number of projects will likely grow quickly into the thousands and millions. The site will be designed to minimize the cost of managing a large number of projects, and to make it possible to locate projects that are useful.

TODO “account represents someone or something who can legally own a copyright”

When a project is created, it is associated with an *account*. A particular user may be associated with more than one account – such as a developer with a personal account and a work account, or a consultant that works with many different accounts – in which case, the user has to specify which account the project is associated with from a *project ownership* point of view. Administration of a project has a number of roles, but fundamentally all administration control flows originally from the project owner.

When a project is created, a number of other attributes are necessary for its configuration, including a combination of its *licensing* and its *access control*. Fundamentally, projects are generally either *open* or *closed*, which means either generally accessible or generally inaccessible. Access control is further subdivided into *read rights* and *commit rights*, allowing projects to control changes such that they can be incorporated into the project only by specific users, which can help to prevent undesired modification. The manner in which access is permitted to the project must be compatible with the license selected for the project, and vice versa. It is minimally required that a set of common licenses be available to select from, with workflow making it easy for a customer to create a project that matches their desired form of IP protection, or conversely their desired scope of distribution.

The selection of the license is important for another reason: It impacts what components and libraries can be incorporated into the project, based on what the licenses are for those components and libraries. For example, a GPL-licensed component should not be used in a project that is licensed out under a BSD-style license or a commercial license. It is important to make these choices simple, and to help customers avoid licensing conflicts that could be introduced via the selection of dependencies. Conversely, it is incredibly important to help a customer find components and libraries that could be useful for their work, and as the number of projects grows, this aspect of curating reusable projects is likely to present a challenge if it is not carefully designed for.

Customers should be able join any *open* project, and if a customer creates or owns a project, that project can be set up to be an open project. Customers can join a *closed* project with permission from the owner of the project. With a combination of complex organizations, employees thereof, contractors thereof, consultants that work across multiple organizations – any of which could also interact with projects as individuals – and individuals, a system for organizing and managing access control for projects will be fairly extensive. Regardless, it must be incredibly simple for a project owner to manage a particular project, while it must be straight-forward for a large, complex organization with many projects and users to manage and monitor those relationships.

(TODO: where does this paragraph go?) A separate collection of related (or derived, or ancillary) open source projects will be hosted and managed by the company, for which the copyright is not held exclusively by the company. These projects may be created outside of the company, and will (like the company’s own projects) indicate one or more licenses that the contributors agree to publish under; the licenses used by the company will be offered as default options, but other licenses can be specified. The reason that a collection of licenses will be presented as default options is that open source licenses can conflict with each other, thus preventing code from one project from being used in another project, or theoretically even preventing code from one project from being used in conjunction with code from another project. As a result, only projects using pre-approved (i.e. fully vetted) licenses will be able to be *curated*, which allows them to be suggested to developers for use in other projects or applications.

When a customer begins to build an application using the site, they are able to make use of the Core IP and any number of curated projects. TODO

Need to explain:

* IP ownership
* Contributor agreements
* Commercial Licenses
* Licensing API versus source versus binary
* Options a customer has for what they can use
* Options a customer has for how they license their work
* Options for how we license things (and why we make those choices)