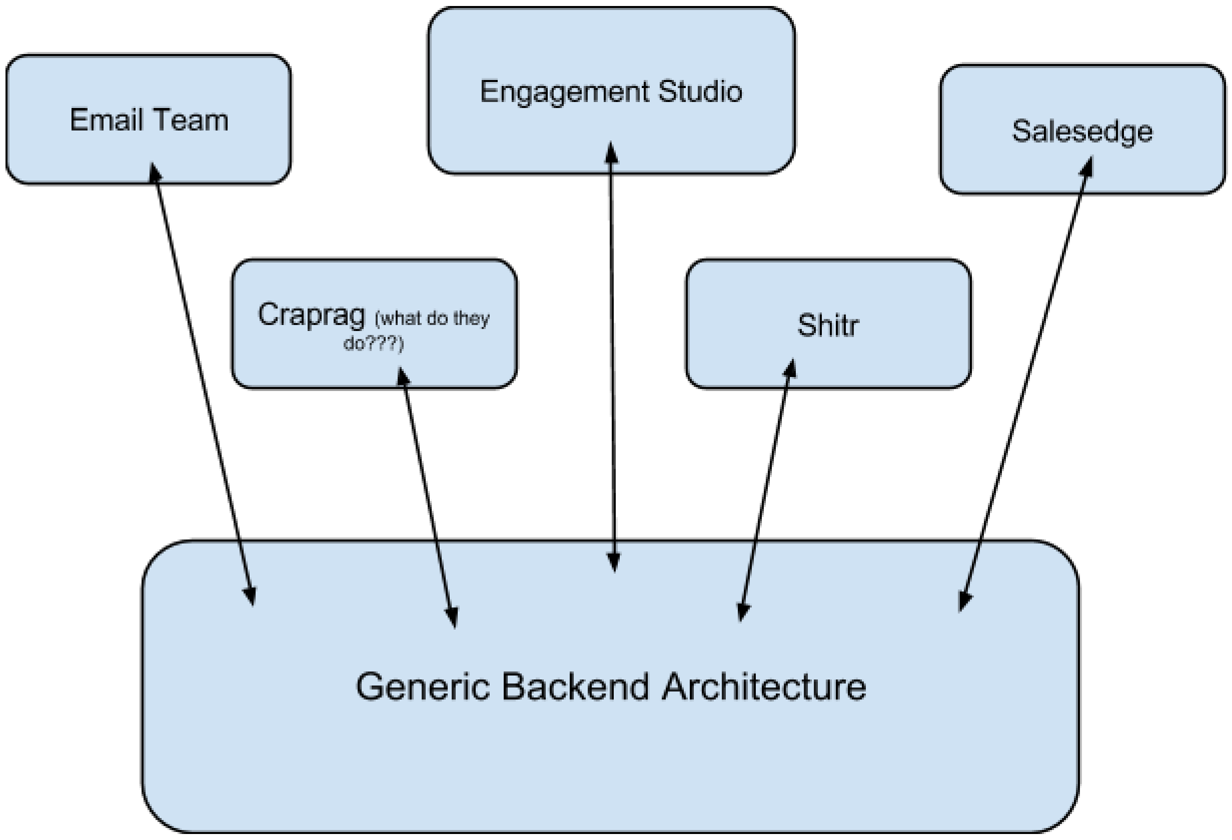
Engagement Studio & Friends

Coming soon to a monitor near you.

Table of Contents

Why the title?

Engagement Studio + Friends is a strange title, and Pardot is an unique company. But the name has a meaning beyond keeping Pardot weird. Originally, team AssLand was going to build Engagement Studio with it's own customized backend that would be hard to integrate with other sectors within Pardot. This strategy would use legacy code that was already written for many workflows. However, after realizing that some extra work would allow a robust and scalable backend that other teams in Pardot could leverage, the team decided to take on the challenge! Therefore, Engagement Studio is just an interface of the newly developed backend that is comprised of three main things (to come later). Hopefully, the backend of Engagement Studio will in fact become leveraged and consumed by many teams within Pardot, making it a generic server-side system.



The next section will talk about how Engagement Studio leveraged this newly designed backend and combined it with a new user interface to meet customers needs.

Engagement Studio

Lead Nurturing in the B2B world drives and educates contacts to certain activities and away from others. It builds relationships with potential clients without them even being aware it is happening. This provides valuable insight into what leads should be contacted today as well as staying on top of tomorrow, giving the Sales department more productivity with less effort. Lead Nurturing is considered the bridge that unites Sales and Marketing departments and it will be the center of Pardot's new marketing automation solution.

Engagement Studio is Pardot's next generation Lead Nurturing product. Engagement Studio is comprised of 3 main themes: Composition, Validation, and Success. Composition tries to leverage marketing automation to create simple and complex nurture programs that resolve a client's problem. Validation then provides confidence to our customers that the engagement programs they have made will succeed. Finally, the Success theme is a metric to gage well the 'success' of each program so that changes can be made to the particular component not meeting expectations.

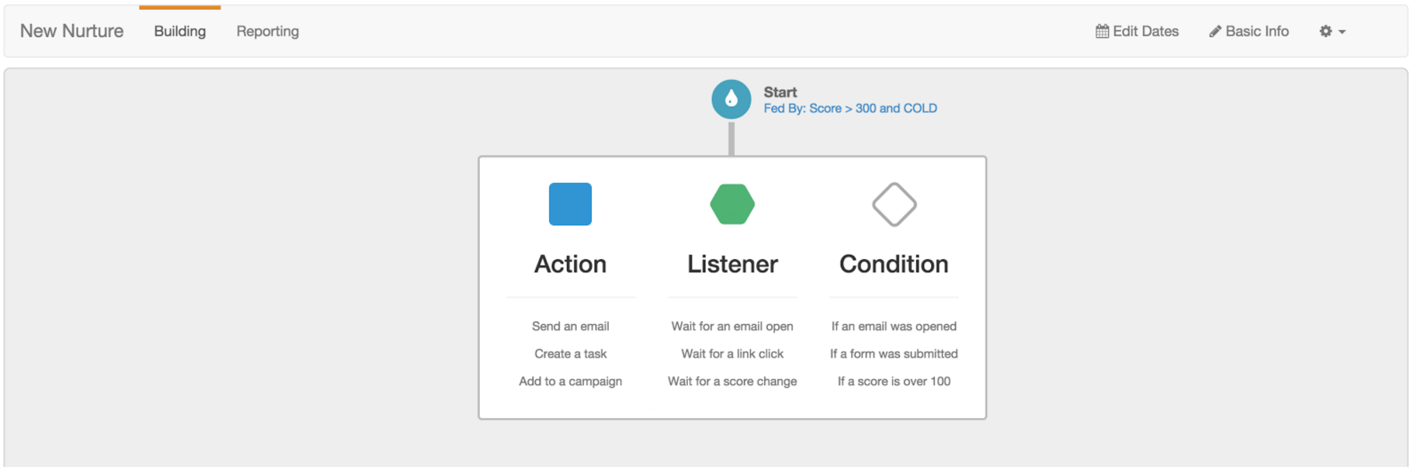
Composition

For the Composition of an engagement program, the design inspiration was taken from the Bart subway map in SF. This intuitive layout also allows users to start an engagement program from a pre-designed template. Everyone has access to the template exchange where a community of leaders, partners, and your own colleagues can upload, comment on, and rate proven successful drips. There are 3 types of nodes that can be used when composing a drip:

Actions are the action the system will take on a Prospect at a given point in time. These actions can be triggered off of a specific date and time or after Listener and Condition nodes.

Listeners literally 'listen' for a specific activity (or even the lack of activity) to occur within or after a certain amount of time. For example, I can set a listener to 'Listen for a specific link in an email to be clicked over the next 5 days'. If a prospect clicks on that specific link on day 2 Engagement Studio will send the Prospect to the next step in the Program.

Conditions are nodes that check specified values and criteria real-time in the system. For instance, if you're trying to qualify a Prospect that exists on a cold lead list you may want to drive that Prospect to a form on your site. If the Prospect completes that form you might want to place a condition node to check for Prospects whose titles are 'Vice President' or 'CMO' and they have a score > 300. These Prospects may then receive targeted content later in your program based off these values.



***Screenshot when a user chooses to make a new engagement program. The user can choose to insert three types of nodes in the drip: Action, Listener, and Conditional.*

Validation

In this step, the user can test out their newly created nurture program to see if the desired actions occur. If an unexpected action takes place in the test, the user can easily modify the engagement program on the fly and start over.

Success

A tool included in each nurture program is a 'health report', which can be found on the main screen. The health report leverages a tool that monitors areas of the drip that are less than performant (i.e. less than 10% of recipients are opening the 'Product Line' email). From here the user can evaluate the report and proceed to carry forward with whatever action makes sense. This setting can be toggled on/off [1].

After seeing how Engagement Studio works from the user's perspective, we can now dive into the architecture of the front and backend to see what's happening under the hood.

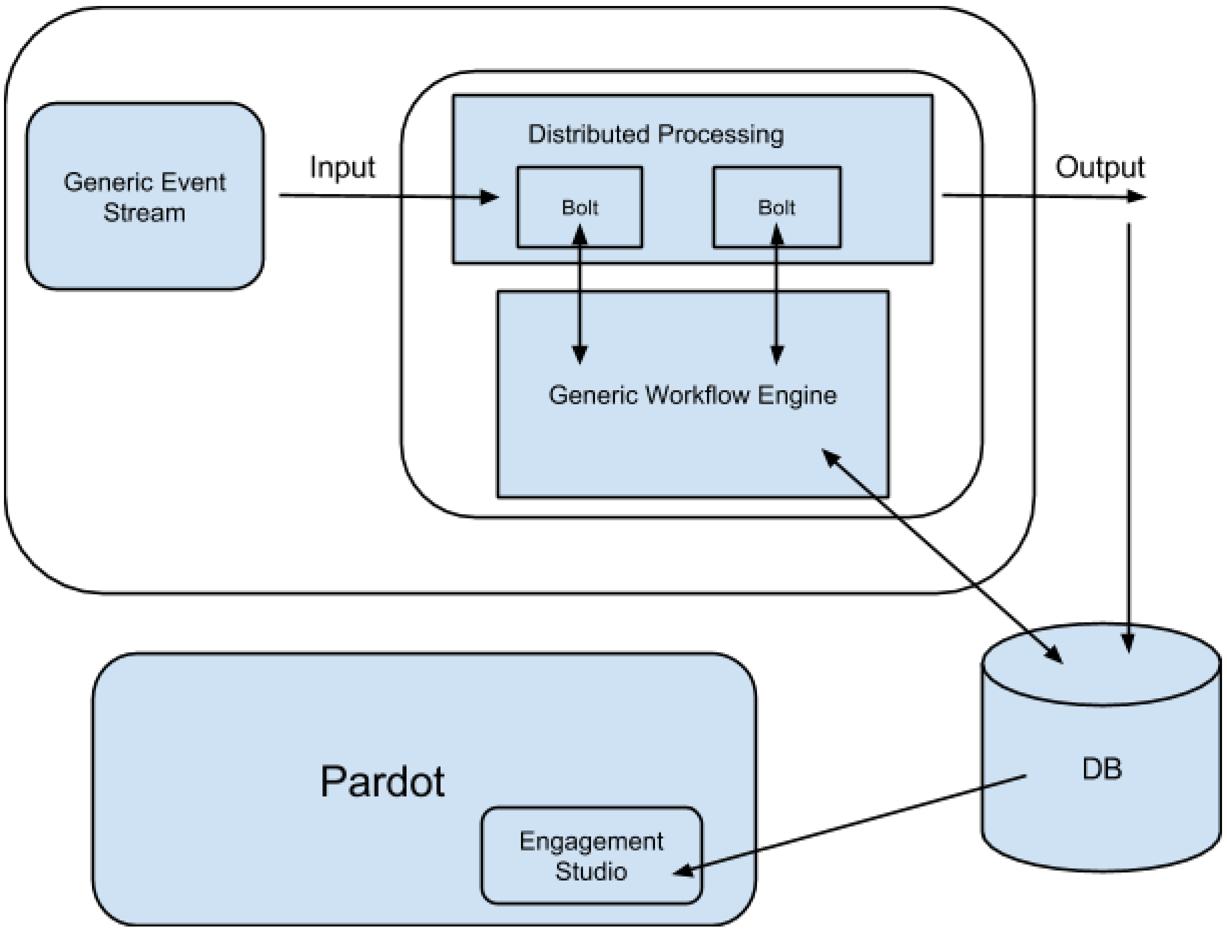
[1] Reference: https://docs.google.com/a/salesforce.com/presentation/d/1LjVaAOjlo9w0rpW_ur9nx5_ADfzZjh2qaIm0aFUWFbE/edit#slide=id.p4

Backend Architecture

As we stated at the beginning of the document, this backend was generically designed and will be leveraged by more than ES. The backend of Engagement Studio decides all of the logic for the workflow assignments to prospects. This has to take into account 4 main things:

1) Generic Event Stream
2) Distributed Processing
3) Generic Rule Engine
4) Engagement Studio

Engagement Studio leverages the first 3 components of the backend for it's own purposes, and displays it to the user interface. Engagement Studio is mainly concerned with engagement programs, whereas this backend will support all types of Workflows: Drips, Automations, Dynamic Lists, and Segmentations. Here is a diagram to convey how these fit together:



Technologies

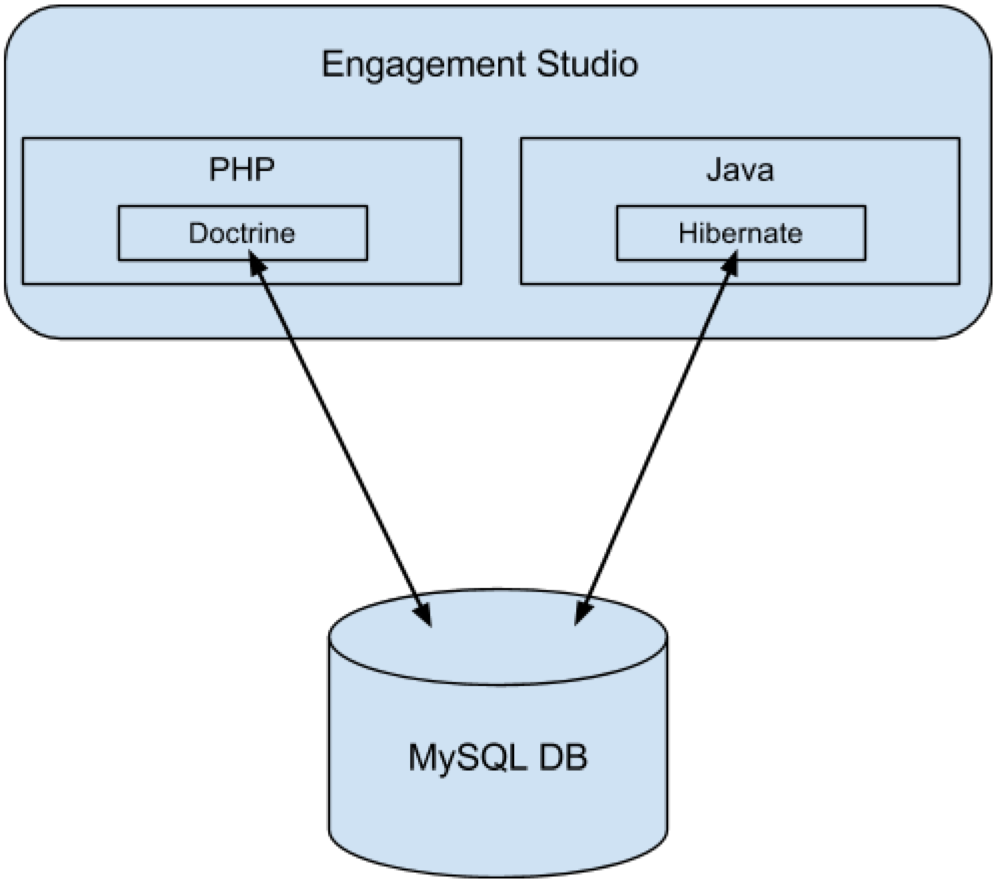
There were numerous technologies in the compilation of the backend for ES. The one’s listed here by no means cover everything that was used, but help break up the major components.

1. MySQL - relational database used to store each instance of the prospect data. Used for the Pardot product as a whole. More information can be found here at http://www.mysql.com/.
2. Doctrine - ORM (Object Relational Mapper) for PHP that connects with MySQL. Doctrine has an easier query language than Propel and is more memory conscious. More information can be found here http://www.doctrine-project.org/.
3. Redis - an advanced NoSQL DB key-value store that can also be used for caching non-persistent storage. Considered a data structure for a server based on how it writes to disk. Mainly used for caching because it is so fast, but has a specified memory capacity. Pardot uses it for caching non-persistent data. More information can be found here: http://redis.io/topics/faq.
4. PHP - driver language for the RESTful API Endpoints used to launch the application of ES and Pardot as a whole. Tons of documentation and information which can be found multiple places, including here: http://php.net/.
5. Kafka - a distributed messaging queue system developed by Apache to be used as a commit log service. Replacing RabbitMQ. More info can be found here: http://kafka.apache.org/.
6. Storm - distributed real-time computation system that does processing on your data. The official documentation can be found here https://storm.apache.org/. It can be configured with any messaging system, database, and programming language.
7. ZooKeeper - handles all server configuration. Mainly used when multiple servers are in deployment and synchronization and naming needs to be mandated between them. Leveraged by both Kafka and Storm to manage server states. More information can be found here: https://zookeeper.apache.org/.
8. Chef - is software that automates your infrastructure. Chef has reusable functions known as Recipes to automate infrastructure tasks. Recipes handle things such as web servers, databases, and load balancers. Chef lets your infrastructure (hardware) become versionable, testable, and reusable through Recipe scripts. More info can be found here https://www.chef.io/chef/.
9. Protocol Buffers - Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. More info can be found here https://developers.google.com/protocol-buffers/.
10. Java - general purpose Object-Oriented programming language that is compiled and run on the JVM. Java was used in ES for all of the backend heavy lifting with Storm. More information can be found here https://docs.oracle.com/javase/tutorial/.

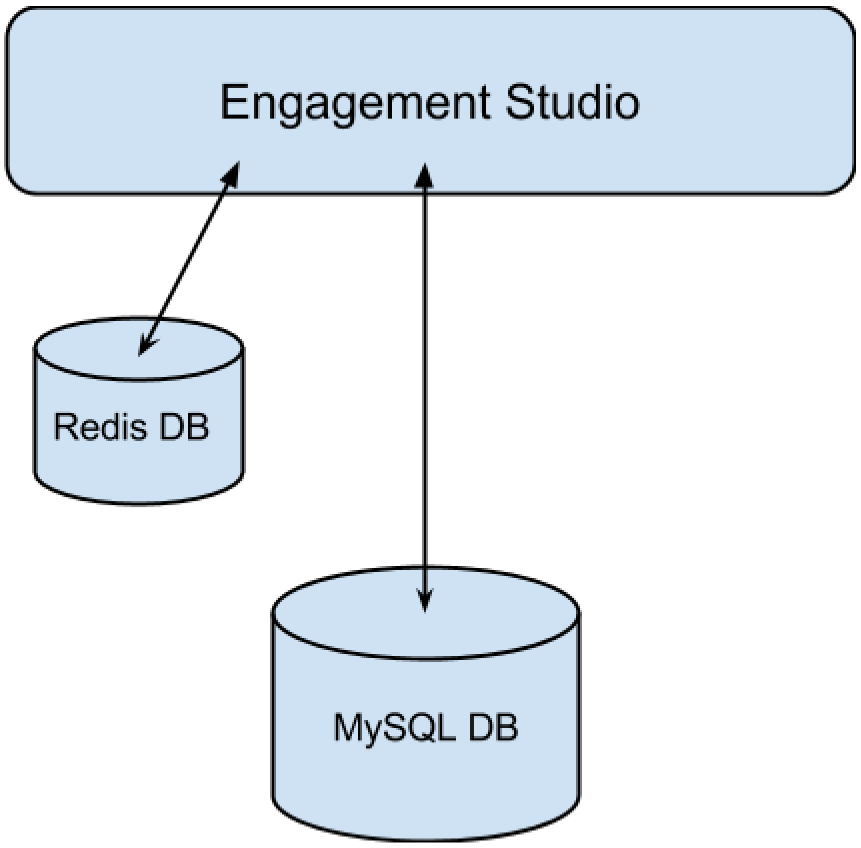
11. Hibernate ORM - ORM for Java that connects with a relational database (MySQL). Hibernate is highly scalable and allows the developer to create persistent classes using Object-Oriented idioms. More information can be found here http://hibernate.org/orm/.
12. PHPUnit - Unit testing framework for PHP, and we have used it to test our RESTful API Endpoints. Allows functionality for mocking objects, methods, and making assertions. Used to test all of the Workflow logic. More information can be found here https://phpunit.de/.
13. JUnit - Unit test framework for Java, and was implemented to test all of the Workflow processing logic in the backend with Storm. Allows much of the same functionality as PHPUnit. More information can be found here http://junit.org/.

Model/DB

Engagement Studio is backed entirely by tables within the MySQL database. The MySQL database is for traditional storage of persistent data (data that will need to outlive the program). PHP is used for only CRUD operations to our RESTful API Endpoints. Java actually handles all of the Workflow processing logic with Storm.



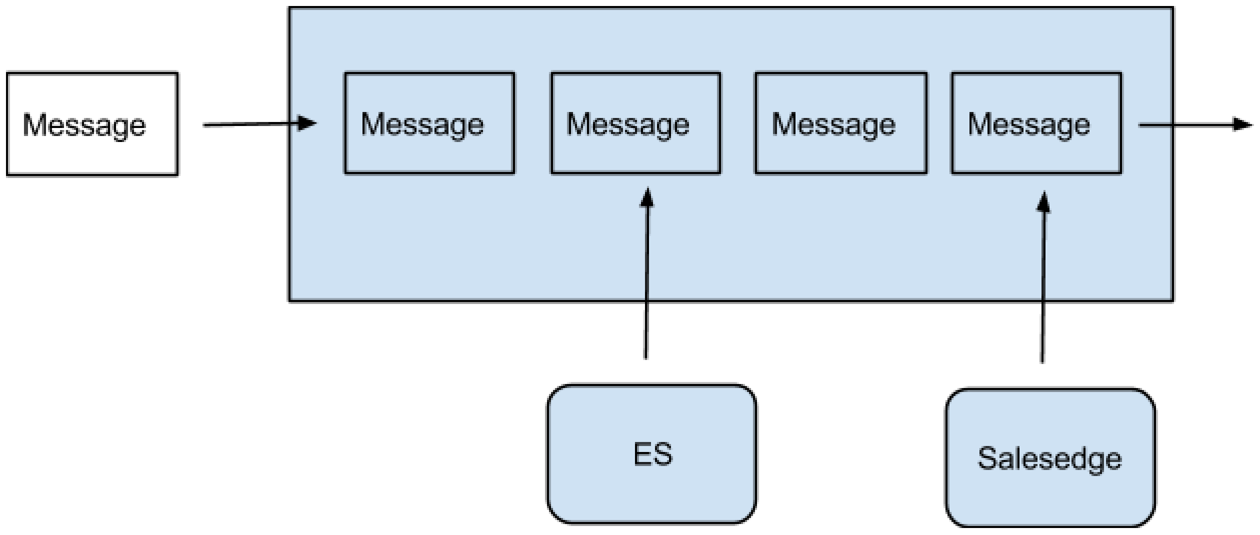
Redis is a key-value store NoSQL database that we use for caching in ES. Redis allows very high read/write speed however with the limitation that data sets cannot be larger than memory, and may or may not be persisted to the disk anywhere. Therefore, ES uses Redis for data that is read heavy to store temporarily in the 'cache', which is also persisted to the MySQL database. Engagement Studio/Client connects to a Redis server using RESP, or their specific TCP connection.



*** AssLand has not defined what parts of the database will be stored in Redis, but we do know that it will serve as a caching layer for some parts of the model.*

Generic Event Stream

The event stream is not a traditional queue, but instead a log file. This generic log file is implemented through Kafka, and standardizes the messages through Protocol Buffers. In Kafka, each application can read the messages in whatever order they desire. Messages are triggered through user actions on the interface. That way, Engagement Studio can have a completely separate pointer to a message in the stream. This will allow the event stream to be generic to many applications throughout Pardot. The messages will live in the stream for two weeks, then Kafka disposes of them to conserve memory. Each application can write their own Protocol Buffer to take any schema of data. Therefore, the JSON (or whatever data objects) need to be formatted to a certain standard in order for Kafka streams to process each message. In this setup, each application can define the schema of their messages through Protocol Buffers, and also consume messages in whatever order they desire.

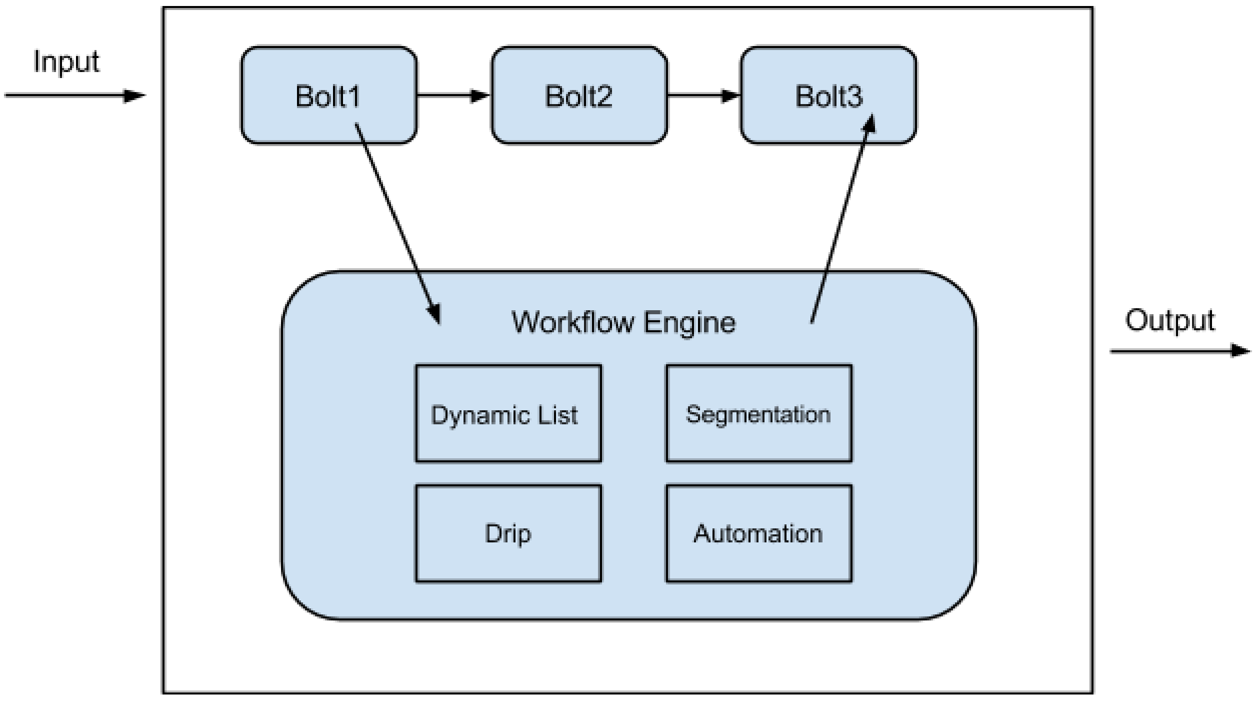


Distributed Processing

We want to be able to run our processes on *n* servers, with very little setup or annoyance. That's where Storm comes in. Storm is a framework for Distributed Processes, specifically message streams, which is exactly what Kafka provides. In this design we are building a topology of spouts and bolts. Spouts are the event streams provided by Kafka that feed Storm log messages. Each of these spouts goes through a series of bolts that we write and design. A bolt can be written in any language and can leverage other code or logic, but Engagement Studio's will be written in Java. Because they are so configurable, many of the bolts in ES spouts reference Workflows of rules and actions that are being rewritten in Java. Bolts have the ability to edit a message, send it somewhere else, or even trigger an action/another bolt. These bolts trigger the logic and functions as a result of user actions. Bolts are the pipeline of actions that decide how to deal with messages they receive from Kafka based on their configuration. To dive into a little more detail, our setup has Storm running messages in 2 stages: the Filter Stage and Processing Stage. In the Filter Stage the bolts are preparing the messages and collecting metadata. In the Processing Stage the data, workflow, and prospect are retrieved and evaluation occurs, which triggers an action. For more information on the spout or bolt configurations, see this doc: https://docs.google.com/a/salesforce.com/document/d/1trQkgOhHU84y4YKXdoAILrFe5WtetACR6rg0S0me6kA/edit.

Generic Workflow Engine

Storm bolts reference the Generic Workflow Engine, where Engagement Studio provides all four features of processing: Drips, Dynamic Lists, Automations, and Segmentations. Each Workflow has it's own set of nodes that perform different actions. The Workflow engine is configured to trigger the rights actions based on what the prospect has done and the type of Workflow defined by the user. For more information on the differences between Workflows check out this site http://www.pardot.com/faqs/automation/prospect-actions/. As bolts read in messages specified by their Protocol Buffer, they can call on the engine to perform actions to prospects. Initially we were going to reference much of the PHP code already written for Rules, Actions, etc. that was being refactored for general use. However, instead of refactoring the legacy PHP code, AssLand has decided to replicate the Workflow Engine in Java. There were many reasons for this, but some included performance, readability, and unifying syntax across the stack. For now this includes all of the Rules processing, which is owned by the AssLand team. However, since we do not own the Action processing code, that will remain in PHP. In the future we hope to eliminate using PHP for our API Endpoints and solely rely on Java for the backend. Re-writing the Workflow Engine in Java will be a huge step to a generic backend architecture that all of Pardot can leverage by simply using the engines API Endpoints. Now any Pardot application can use Kafka messages and their own bolts to update prospect activity. They simply just need to attach their own user interface to this backend. Here is a diagram of how they fit together:

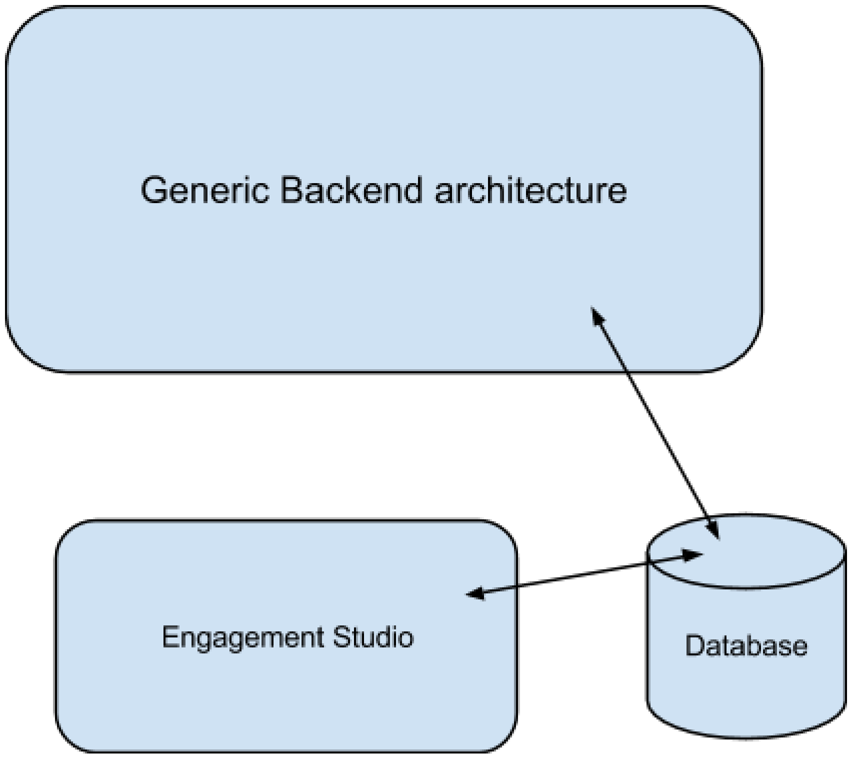


Engagement Studio

Finally, to wrap up the backend, Engagement Studio leverages the first three generic components to meet it's purpose. Engagement Studio configured it's own Protocol Buffers to fit the standard for it's messages in Kafka. Then it retrieves the messages through it's pointer and funnels them through Storm spouts, where they await bolts. These bolts are configured to do a number of things, and are interconnected with the Workflow Engine. Engagement Studio is a drip program interface, therefore it uses the Drip workflow which is composed of:

1) Action Node
2) Rule Node
3) Branch Node
4) Sleep Node
5) Start Node
6) End Node
7) Timeout Node

These nodes are self explanatory, but each trigger a different action and are associated with different rules. The engagement programs that are created in the user interface are then modeled on the backend consisting of these nodes. Prospects are then run against these programs created by the user. Here is a diagram ES and the general backend:



Hopefully the backend architecture isn’t black magic anymore. Engagement Studio can simply leverage and configure Kafka, Protocol Buffers, and Storm spouts/bolts to use the parts of the general backend that it needs.

Conclusion

What started as a project to build the most amazing engagement program on the planet has actually turned into a generalized backend that will benefit many teams within Pardot. Not only will this design benefit Engagement Studio in the future, but also many other teams that leverage the Workflow engine, and message logging. As Pardot continually grows as an organization systems that are general and easy to configure will become invaluable.

Frontend Architecture

The frontend of Engagement Studio is a much less complicated system then the backend. However, there are still many components that come together to bring the interface into life. The main framework used on the frontend is Ember.js, which is very robust and comes with many out of the box features that makes it easy to set up a scalable web application.

Technologies

1. HTML - no need for explanation here. It's the markup of the web.
2. Handlebars - templates that compile into HTML. Great for semantic markup and allow more powerful expressions like variable inclusion. Handlebars allow live bindings of variables, which automatically tracks updates and displays the changes in real-time. More can be found here: http://handlebarsjs.com/.
3. CSS - no explanation needed here. Styling of the web.
    a. Bootstrap - library released by Twitter that has been built in code for specific styling of components. Uniform way to style web components such as buttons, forms, lists, etc. More info can be found here http://getbootstrap.com/. Note this link is to Bootstrap 3.
    b. Less - a CSS pre-processor that extends the language and allows it to be more maintainable, extendable, and reusable. More info can be found here http://lesscss.org/.
    c. Font-Awesome - a library of pictographic icons for easy scalable vector graphics on websites. These icons can be customized and easily inserted and reused similar to Bootstrap. More info can be found here http://fortawesome.github.io/Font-Awesome/.
4. Bower - package manager for client-side assets. Runs on top of node.js and is leveraged through ember-cli. Handles libraries such as D3, font-awesome, and bootstrap. More information can be found here http://bower.io/.
5. Node.js - a server side solution for JS. What powers the ember-cli server and holds any server-side JS code. Also comes with a robust package manager. More info can be found here https://nodejs.org/.
6. NPM - the package manager for node. Used to get ember, ember-cli and all of it's dependencies that it can manage, broccoli, etc. Has great documentation and even allows one to manage a private NPM repo through Sinopia or NPM Enterprise. More info can be found here https://www.npmjs.com/.
7. Ember Framework
    a. Ember.js - 'A framework for creating ambitious web applications'. The main framework of the client-side of ES, and comes with many common idioms so you spend more time writing code instead of configuring structure. Favors

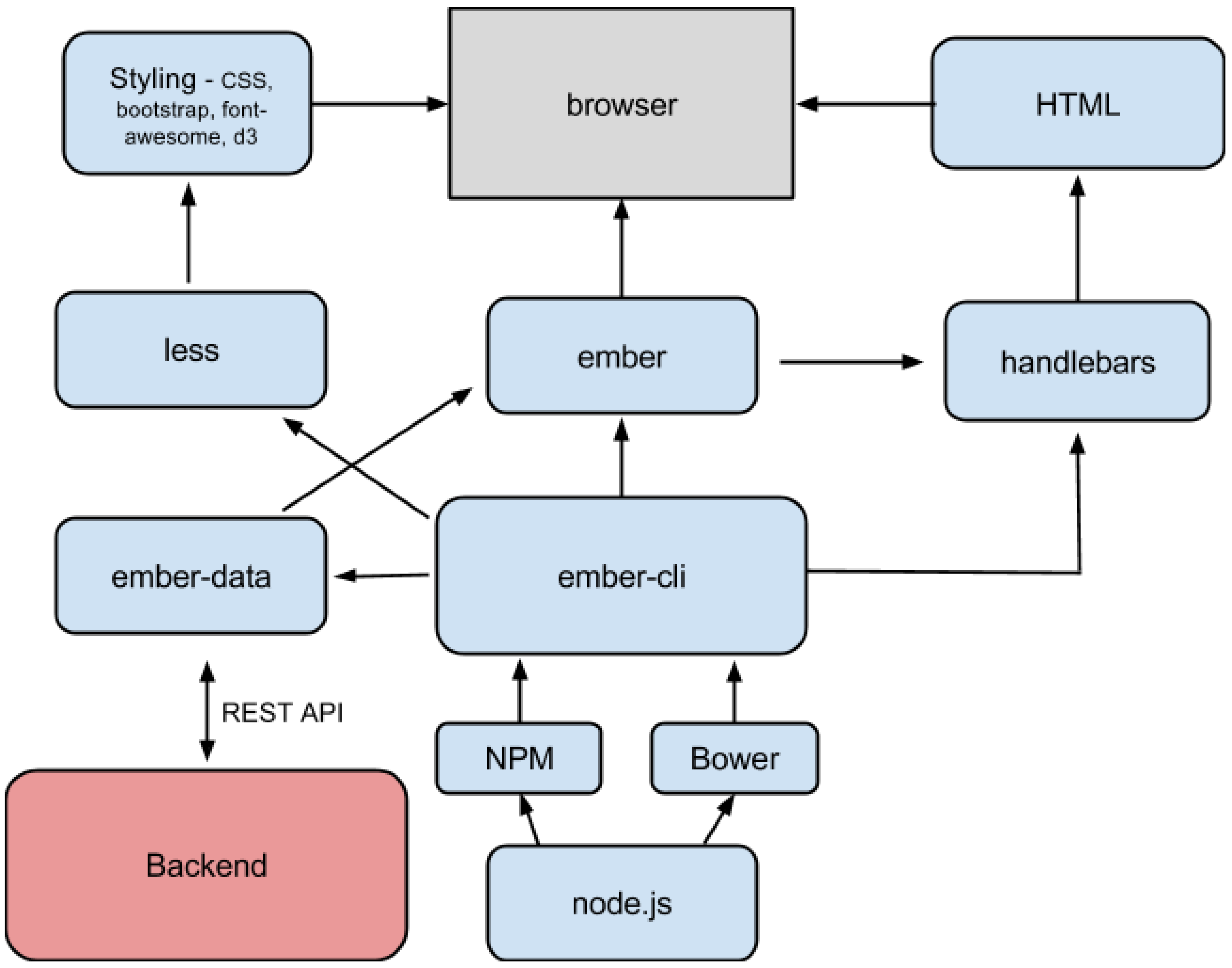
convention over configuration. Has an amazing data model layer, ember-data. More information can be found here http://emberjs.com/.

   b. Ember-CLI - a command line application for Ember apps that runs on node.js. Allows you to run your application and asset management as it leverages Broccoli.js. Come with many built-in features and allows convention over configuration. More information can be found here http://www.ember-cli.com/.
   c. Ember-Data - a library apart of Ember that manages the Model data on the frontend. Makes it easy to retrieve records from a server, cache them for performance, create new records on the client, and save them back to the server. Uses a JSON RESTful API by default. More great documentation on this aspect check out this page http://emberjs.com/guides/models/.
   d. jQuery - well know JS library that has an API for DOM manipulation, event handling, animations, and AJAX calls. More information can be found here https://jquery.com/.

8. D3 - JS library that helps you provide dynamic data visualizations in web browsers. Provides powerful visualization components using HTML, SVG, and CSS. **D3** allows you to bind arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, you can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction. More info can be found here http://d3js.org/.
9. Testing
   a. SinonJS - a JS testing library that is great for unit tests. Allows you to create spies, stubs, and mocks. Stubs are just spies with functional behavior that the user can define. Mocks include fake functions, objects, HTTP requests, and even a server. More info can be found here http://sinonjs.org/docs/#mocks.
   b. PhantomJS - a JS framework for a headless browser that is great for testing and has a built in JS API. This is great as it can test any JS code and also screen capture, networking, and page automation. More info can be found here http://phantomjs.org/.
   c. QUnit - JS unit testing framework. Library that allows one to unit test on the client-side. Gives the programmer confidence to write JS code in function and not inline handlers. More information can be found here http://qunitjs.com/intro/.

Technologies Diagram

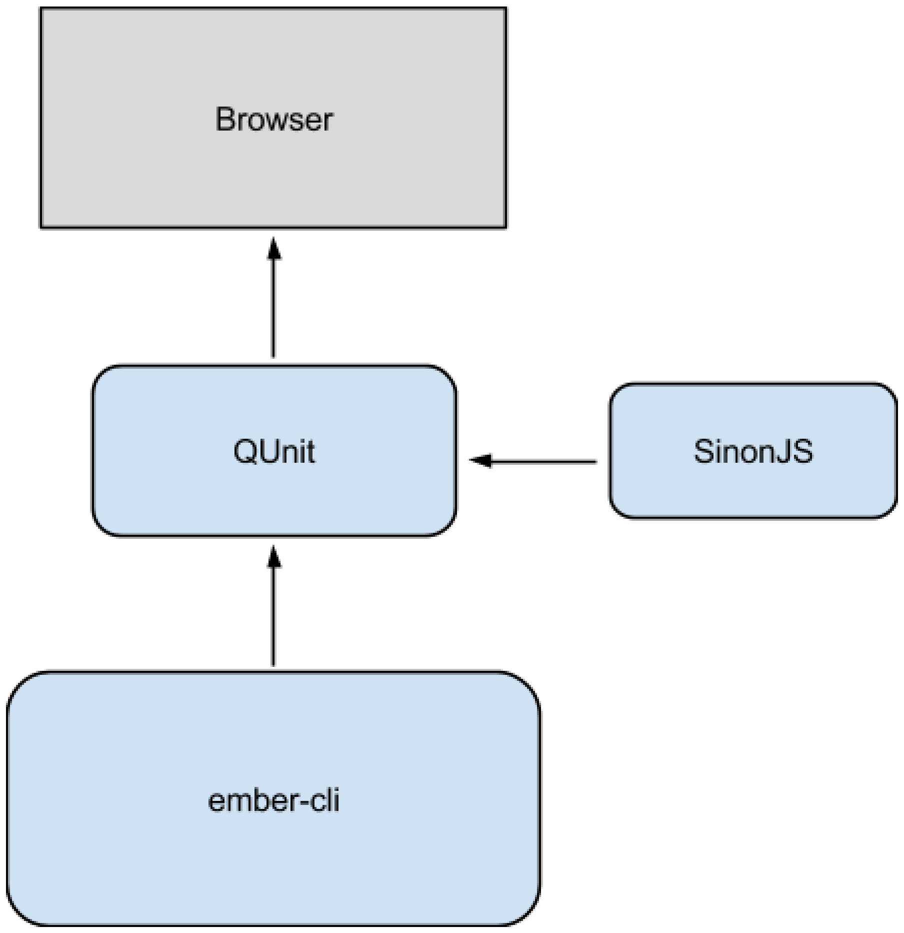
So there's a lot of technologies used to build the interface of Engagement Studio. However these components do come together to make an awesome drip program. Below you will see a diagram of how they are interconnected. Starting from the bottom, node.js is needed to run ember-cli. Because ember-cli is server-side and is written in JS, node is the only JS solution that can be run through a server. Ember-cli does many things, but comes with ember-data, a library tightly integrated with Ember that is used to easily persist data to the server and manipulate models. Ember-data doesn't have to be used with the ember framework (could use jQuery) but comes with robust functionality that makes model manipulation on the front-end much easier. Ember-data is connected to our backend and communicates to it through a RESTful API. Ember-cli is responsible for compiling handlebar templates into HTML which is rendered by the browser. Lastly, ember-cli compiles less files into CSS for the browser.



The Ember framework requires handlebars to render templates, and handles all of the JS used in the functionality of Engagement Studio. Ember holds the driver file app.js, and many others including the routers, controllers, and adapters. All of the packages our Ember app uses are stored in the node_modules directory which is connected to NPM, a package library that comes automatically with node.js. Package.json handles the server-side dependencies from our app to NPM, while bower.json handles the client-side assets for Bower. Finally, all of our styling for each page is written in less, and we leverage open-source libraries like bootstrap and font-awesome. Bootstrap is pre-written structured CSS that is interjectable into handlebar templates or HTML. Font-Awesome is scalar vector icons that is also interjectable into handlebar templates or HTML, and is known for rendering quickly. Less is more readable markup that is compiled back into CSS by ember-cli for the browser. All of these components come together to make a scalable web application leveraging many open-source libraries.

Testing

Below is a diagram of the testing architecture used for ES. QUnit is the unit testing framework run through the browser to test JS logic on the DOM. It leverages SinonJS for mocking objects, functions, and even fake HTTP requests. Ember-cli runs the testing suite through it’s internal server.



Agile Development of ES

Lastly, I wanted to touch upon the process of building the interface of ES. The AssLand team has a Project Manager (Kyle Coleman) and a UX designer (Cliff Seal). Cliff is responsible for working with Kyle on what the customer needs, and drafting mockups for potential pages and layouts of ES. Then, Kyle conveys to the rest of the team (who are way cooler) long term goals and directions for the interface. These goals are then broken down into tickets by Eric Thomas and Ben Grimes (AssLand's front-end engineers). Tickets are modular components that could be anything from coding to spikes. Finally, these tickets are picked up by engineers in sprints that last two weeks. The beauty about this process is that because everything is done in small increments and very quickly, changes to the UI or vision of the project can happen without major consequences.

Conclusion

Engagement Studio's interface will be a complete revamping of our current drip programs and give the user a cleaner and more powerful interface. Even though many technologies are used, a powerful framework such as Ember (with ember-cli) and an intuitive package manager keep the project bearable to manage. For more information on the status of ES or the direction of the project contact:

Eric Berg - AssLand engineering manager
Kyle Coleman - AssLand PM
Eric Thomas - Frontend engineering lead
Stevie P - Backend engineering lead
Brad Ware - *extremely handsome intern

*Voted unanimously by the Pardot office