Project documentation

Play-Pin

Ciobica Elena Alexandra S3257061
Isac Flavius Andrei S3257053

1. Introduction

The purpose of this document is to provide documentation regarding Play-Pin application that is an image aggregator. The document will describe the scope of the project, architecture, technologies used and evaluation of the project.

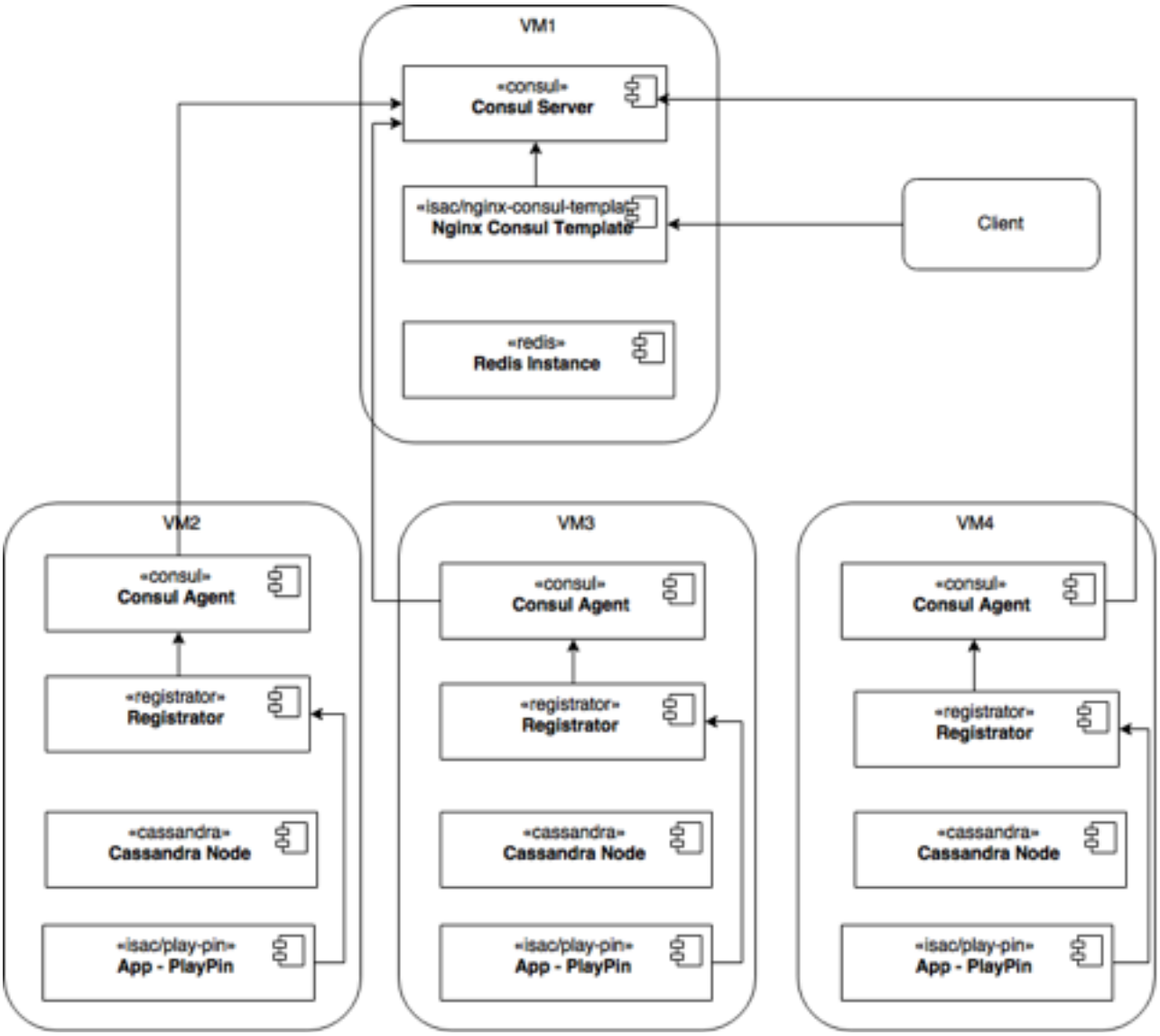
2. Scope

The project consists of a scalable, fault tolerant web application designed for image sharing. Play-Pin was developed to the current stage during two months. The application does not require registration from the user thus, anybody can use the service without logging in.

Functionalities:

- Create new boards (feeds), that will be available to all the users of the application
- Upload pictures to specific boards
- Browse through the entries of a board — the users that are browsing pictures on a specific board will get a live notification if a photo is uploaded to that specific board, so the boards are updated dynamically once a new entry is

3. Architecture - global



The project architecture was designed using **4 virtual machines** with a fault tolerance of loosing one machine. Furthermore, the goal is to not use any hardcoded ip. Due to hardware limitations on the development computers (4 threads, 8gb ram) no more than 4 virtual machines with 1 gb ram each could be used, also due to the low amount of credit on platform such as AWS (55$) or DigitalOcean (50$) the deployment of the project will be done on a cloud computing platform only before demo date.

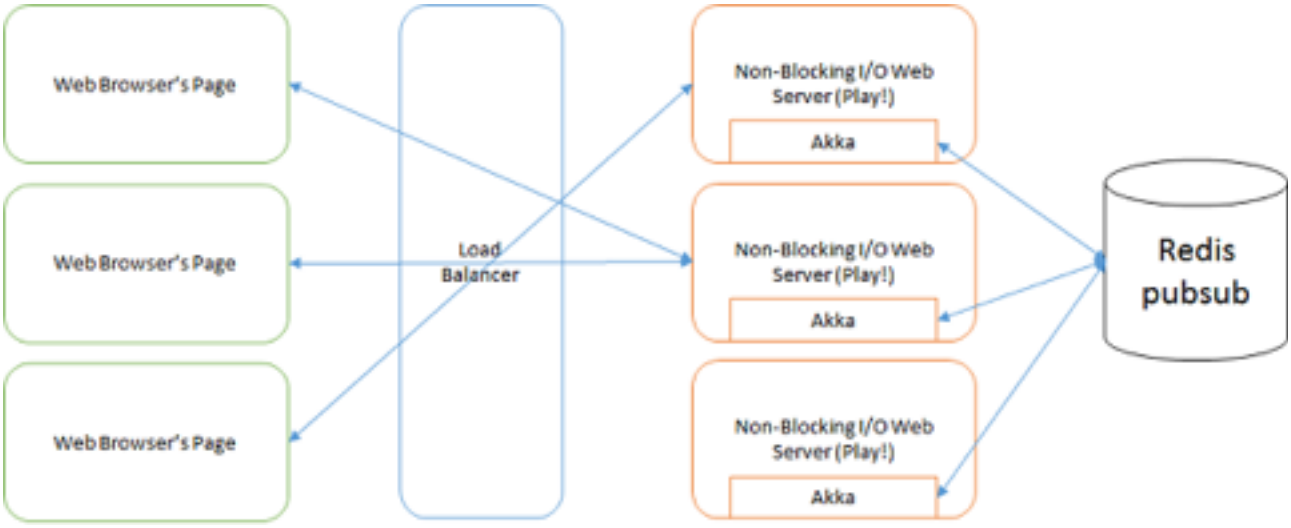
The App - PlayPin container is where the main application is deployed. The application is **developed in Scala** with the Play Framework. It is deployed across **3 virtual machines** for fault tolerance and user distribution. Is is also **run inside the weave network** in order to have access to the Cassandra and Redis instances.

3.1 Databases

- **Cassandra** is used for storing the information (boards and pins), it is deployed across 3 virtual machines, forming a 3 nodes cluster. The data will be replicated twice, so for every write request the cluster distributes the data in two nodes. This allow us to have consistency even if one node fails, but also requires a lower disk space on each node (ideally 2/3 of total data). **Phantom DSL** is used as a plugin designed for Scala and Play. The consistency on write is set to ALL, which means it checks if the two writes are successful and the consistency in reads is set to ONE, which result in a full consistency model. Since we do not have a low of writes (maybe 10:1) it is ok to set the consistency to ALL on writes and we do not need to go for “eventual consistency”.

- **Consul** is used together with Registrator and Consul Template, Consul Template running inside the same container as Nginx. This allows us to automatically register services to Consul, generate the Nginx configuration with Consul template and restart Nginx every time there is a change in the running containers of the application. Consul can also be used for registering Cassandra or Redis, but issues appeared during development (further explained in the Issues/ Evolution section).

- **Redis** is used for its Publish/Subscribe messaging paradigm it implements. This allow us to create scalable Web Sockets communication:

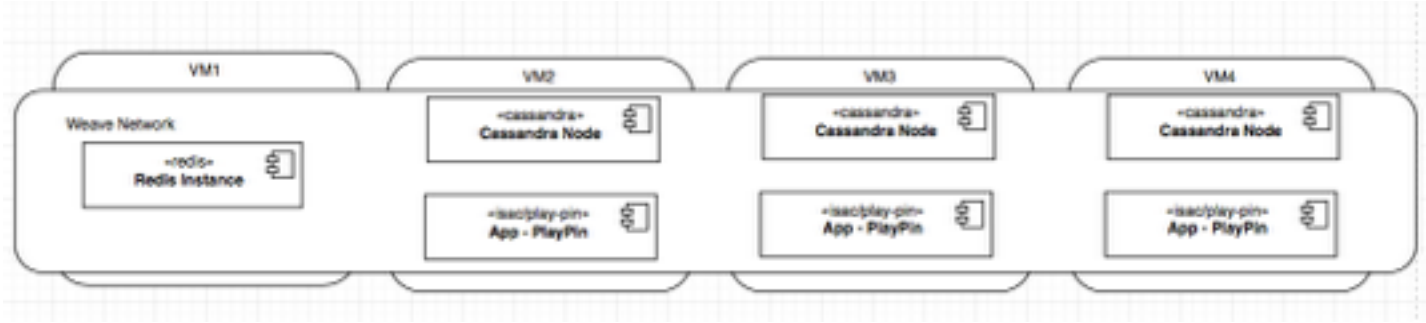


The initial architecture assumption was that Cassandra allows placing a trigger every time a successful write is done, and in theory it is possible and supported, but no or little documentation is available, it also implied that Cassandra image should have been altered because Phantom has no support for such a feature. By dropping this assumption, initially, the publishing was done from the front-end, but this created the problem that only users that are on the same server would get notified.

The solution was to use the Redis PubSub features, with the “consul-api” plugin, which supports by default registration of Akka Actors. Also, in this way multiple channels(boards) are very easy to implement.

3.2 Communication and Security

A Weave network is used to connect Redis, Cassandra and App Play-Pin. This automatically solves some security issues, because Redis and Cassandra are not available outside the network, and the application is only exposing port 8080 for HTTP requests.



This leads to failure of communication between the consul agents and the application, therefore consul Key/Value storage cannot be used at the moment, this will be further explained in the Issues/Evolution section.

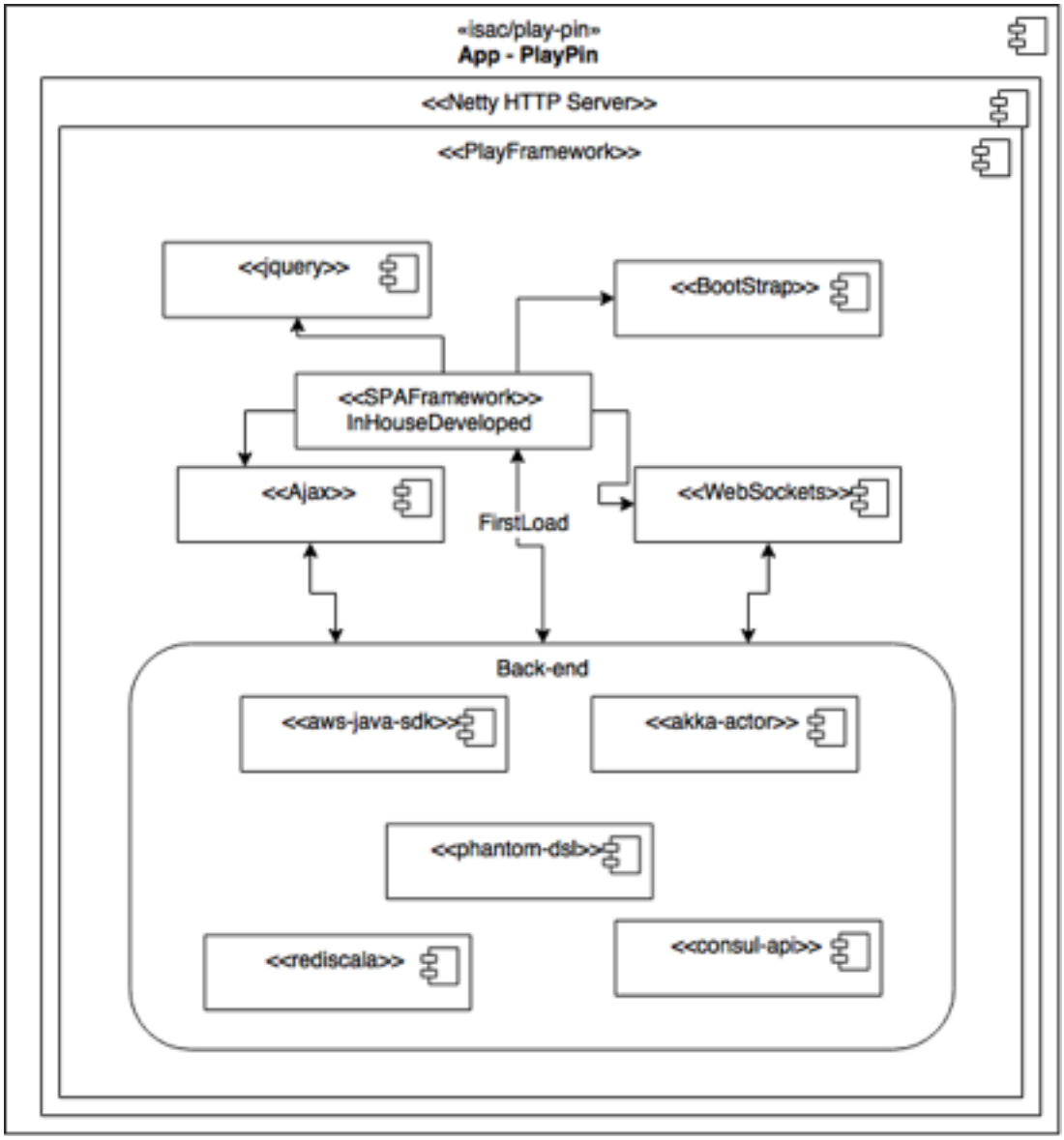
3.3 Virtual machines deployment

The deployment of the VMs is automated using Vagrant.

3.4 Image storing

Amazon S3 is used to store the images.

4. Architecture - app



- The application is developed in Scala, using the Play Framework. This configuration comes along with using Netty HTTP server and SBT package manager. Netty is just another HTTP Server and it does its job just fine. SBT plays two roles in the development and deployment part of the project. It includes all dependencies needed by the project and has the ability to publish the project directly in a docker image, which is very handy as there is no need to write and build the docker image manually.

- A simple **SPA** framework was developed in jQuery using Ajax instead of using an existing framework such as AngularJS. **Bootstrap** framework was used to design the pages.

- **WebSockets** provide the channel to push notifications and content from the server-side to the clients.

- **Amazon Web Services** Java SDK is used to store the new images inside the Amazon S3 Bucket. The image link is generated and the information about the new entries is stored in Cassandra using Phantom DSL plugin.

- **Akka Actors** are used to subscribe users to channels(boards) in order to receive live notifications through WebSockets when a new image was uploaded in the respective board. The actor is subscribed to the Redis Pub/Sub through the RedisScala plugin.

- **Consul API** will be used to interact with consul Key/Value Storage.

The application is **fully asynchronous**, all connections to Cassandra and Redis are asynchronous. This was implemented by choosing plugins that support asynchronous connections, both Phantom DSL and RedisScala support it by default. Unfortunately, Consul API is not asynchronous, but since it will be used just when the connection to the Cassandra is initiated (once per server) it is not an issue.

5. Technologies and Justifications

5.1 Scala & Play & SBT & Docker

- Scala was chosen at first for the bonus points, the lack of documentation was a big issue in using this technology, but compensated with the easiness of writing asynchronous operations.

- Play framework was a natural choice, as it is the best Scala framework for web applications at the moment.

- SBT is integrated in the Play framework and is the quickest way to include dependencies and deploy the project into a docker image.

- Docker was used to containerise the application for the easiness of deployment across different hosts.

5.2 Vagrant

Vagrant was used for the auto-deployment of the virtual machines. It was preferred instead of using something like Docker Machine or Docker Swarm Mode because it gives the developer full control. You know at any time where and which container runs and this allowed us to get a better understanding of how things should work.

5.3 Weave + Weave DNS

Weave was used to create an overlay network across our containers, as well as to map DNS addresses to Redis and Cassandra nodes. Since Docker 1.9 was released, Weave is no longer required in order to have an overlay network across different virtual machines or data centres. Docker 1.9 has default support for overlay networks. Still, Weave is easier to setup and can map DNS addresses to containers inside the network.

5.4 Cassandra

Cassandra was chosen for data storage. At first we have chosen MongoDb, but after the database course we have switched to Cassandra because it is a better fit for you project. Our data consists mainly of image feeds, which are time-based series, an area where Cassandra performs best.

Two tables are used, one for the boards, containing just the name of the boards, and one for the pins. The pin table has as partition key the board name and a primary key as the unique UUID of the pin. This allows a good distribution of data across all the nodes and also keeps all related information close together, on a partition key. Cassandra also supports pagination, so we can avoid loading all the pins from a board at once, unfortunately Phantom has a issue with this feature.

5.5 Consul + Consul template, Nginx, Registrator

Consul was chosen for service discovery and key/value storage. Zookeeper and Etcd were considered too, a few years ago Zookeeper would have been the obvious choice because Phantom had default integration with Zookeeper (deprecated now). Consul was chosen over Etcd because it brings together tools that with Etcd you would have to install separately. Consul has its own interface, health checking system, default support for Registrator and Consul Template. Registrator was used for auto registration of the services into consul. A new docker image was built for Consul template and Nginx, where Consul template listens for changes in Consul and generates the Nginx configuration file every time there is a change, restarting Nginx in the process. HAproxy was considered as an alternative for Nginx, but Nginx was chosen for the ease of use and previous experience.

5.6 Redis - Redis Pub/Sub & Web-Sockets

Redis was chosen for its Pub/Sub feature because its the most easily integrable component. We needed a way to subscribe/unsubscribe actors for a channel and we had 3 options on this:

1. Build our own Akka Actor System to handle this.
2. Use a queue such as RabbitMq
3. Use Redis Pub/Sub

Redis came out as a winner besides the fact that it is not probably the best solution, but is the easiest one to implement, works perfectly and requires almost no time to setup. Redis does not need to be replicated as it is not a critical component of our system, if Redis fails, the application still works, except the notifications for new pins.

The Subscribe Mechanism works with Web-Sockets, when the user opens a board page, a new Web-Sockets connection is opened and he is automatically subscribed to that channel in Redis. When another user successfully uploads a new pin to that board the server is notified through a POST request, which publish the message to the board channel in Redis and all the Subscribed users will be notified through the already opened Web-Sockets connection.

5.7 SPA - own framework & Bootstrap & jQuery

We have chosen to develop our own SPA framework due to time limitations. Angular JS was considered, but we considered that the time needed to accommodate with the framework would be better used by developing a better and fault tolerant back-end.

Our SPA framework is really simple, all it does is transform every <a> in the page into an ajax call, and replace the “content” with the response of the ajax.

Of course, Bootstrap with a custom template was used to design the template and Jquery was used to ease the work with javascript.

6. Evaluation (Reflections+ Issues + Improvements)

6.1 Reflections

There was too little time to implement the project with all the ideas we had, deliver the expected high code quality and try all the new technologies.

We focused on:
- making a working application
- easily scalable
- with no hardcoded ip.

The course was too short and covered a lot of ground really fast, we had to solve some problems ourself and we did not always we find the required documentation/answers online, sometime bugs or missing functionalities arise in the used plugins. But in the end, we are pleased with the result.

6.2 Issues

1. Because Phantom and Cassandra does not have support for triggers, so we had no way to trigger an event on all the servers after a successful write in the database. This means that the application fails to notify that are on a different server that the one on which he upload is made. To solve this we use Redis Pub/Sub, subscribe all the users and publish the message to Redis which will take care of the notifications. [solved]
2. Cassandra pagination is implemented in the Phantom plugin, but there is an issue currently in the implementation (even posted on github), so there is no way to get the paginationState Cassandra needs to continue the last query. Right now we just get all the images in a board. [unsolved]
3. We did not manage to run Consul inside Weave, or map its DNS port to 53 (again an issue is opened on consul github page), so currently we do not use consul for its key/value storage. Explanation: Consul can be access by localhost if it runs on the same network, or by consul.service.consul if the DNS is reachable from inside the container [in progress]

6.3 Improvements

1. Adding distributes performance tests (Gatling or JMeter).
2. Improve code quality and remove duplicates and commented code (please check the code source after the weekend to see nice code)
3. Migrate to Angular 2.
4. Design improvements, pin page, add pin button on board page.
5. Add graph database and randomly add relations between pins. Show the first few grades of friendship related pins on the pin page.
6. Find a way to properly find relations between pins (probably a little hard using Cassandra, maybe an indexing component is needed, such as Apache Solr).