**دانشگاه صنعتی امیرکبیر**

**(پلی تکنیک تهران)**

**دانشکده مهندسی کامپیوتر و فن‌آوری اطلاعات**

**پایان‌نامه کارشناسی**

**گرایش نرم‌افزار**

**عنوان**

# پیاده‌سازی بستری برای اینترنت اشیا

**نگارش**

**پرهام الوانی**

**استاد راهنما**

**دکتر مسعود صبائی**

**آبان ۱۳۹۶**

به نام خدا

تعهدنامه اصالت اثر

تاریخ:

اینجانب پرهام الوانی متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید دانشگاه صنعتی امیرکبیر بوده و به دستاوردهای دیگران که در این پژوهش از آنها استفاده شده است، مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارائه نگردیده است.

در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

کلیه نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه صنعتی امیرکبیر می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه صنعتی امیرکبیر ممنوع است. نقل مطالب با ذکر مآخذ بلامانع است.

پرهام الوانی

امضا

## تقدیر و تشکر:

از پدر و مادرم که همواره در زندگی راهنما و مشوق من بودند و بدون یاری آن‌ها به پایان رساندن این دوران هرگز ممکن نبود.

از جناب آقای دکتر صبائی بابت کمک‌ها و راهنمایی‌هایشان که در نهایت این پروژه را حاصل شد کمال تشکر را دارم، زیرا بدون کمک‌های ایشان هرگز اینترنت اشیا در این دانشکده شکل نمی‌گرفت و این پروژه هر چند کوچک از آن حاصل نمی‌شد.

از اساتید محترم جناب آقایان دکتر فلاح و دکتر نورالحسینی که زحمت داوری این پایان‌نامه را بر عهده داشتند نیز قدردانی می‌کنم.

در انتها از تک تک دوستان خوبم که بنده را در پیاده‌سازی این پلتفرم یاری کردند از جمله آقایان ایمان تبریزیان و محمد حسین توکلی بینا کمال تشکر را دارم.

# چکیده

در اکو سیستم اینترنت اشیا، هوشمند بودن اشیا شرط لازم است ولی تنها فاکتور تعیین کننده نیست. اشیا هوشمند حجم بسیار بالایی از داده را از محیط حس می‌کنند. قدرت و ارزش اصلی راهکارهای مبتنی بر اینترنت اشیا در این نکته نهفته است که به چه نحوی بتوانند این داده‌ها را جمع‌آوری، تحلیل و بر اساس آن تصمیم‌سازی کنند. در معماری اینترنت اشیا، این مجموعه از وظایف بر عهده‌ی لایه‌ی پلتفرم است.

پلتفرم اینترنت اشیا یک بستر برای مدیریت اشیا، جمع‌آوری داده‌ها و تحلیل آن‌ها است. معماری این بسترها از آنجا که تحت بار زیادی قرار می‌گیرند یک نکته‌ی حائز اهمیت در موضوع پیاده‌سازی آن‌ها می‌باشد زیرا می‌بایست بتوانند خطاپذیری و مقیاس‌پذیری بالا را برای بستر فراهم آورند.

معماری ریزسرویس[1] یک معماری جدید برای پیاده‌سازی سیستم‌هایی می‌باشد که در آن‌ها مقایس‌پذیری و خطاپذیری اهمیت به سزایی دارد. در اینجا به نحوه‌ی پیاده‌سازی پلفترم اینترنت اشیا با معماری ریزسرویس‌ها می‌پردازیم.

## واژه‌های کلیدی:

اینترنت اشیا؛ پلتفرم اینترنت اشیا؛ معماری ریزسرویس

[1] Microservice

صفحه

## فهرست عناوین

## فهرست جداول

صفحه

## فهرست اشکال

صفحه

# ۱

# فصل اول

## مقدمه

## مقدمه

هنگامی که کاربران در ساختمان‌های اداری، مسکونی و تجاری توانستند از طریق اتصالات اینترنت باسیم با کاربران دیگر ارتباط برقرار کنند انقلابی در زمینه ارتباطات رخ داد. موج دوم این انقلاب زمانی بود که تجهیزات سیار مصرف کنندگان (لپ‌تاپ‌ها، گوشی‌های تلفن همراه هوشمند، تبلت‌ها) از طریق اتصالات بی‌سیم به یکدیگر و شبکه‌های عمومی متصل شدند؛ اما موج آخر مربوط به اتصال اشیا به کاربران، شبکه‌های تجاری و عمومی و دیگر اشیا از طریق ترکیبی از اتصال به اینترنت است. این موج اتومبیل‌ها، هواپیماها، تجهیزات پزشکی، آسیاب‌های بادی، حسگرهای محیطی، تجهیزات استخراج گاز طبیعی و بسیاری دیگر از دستگاه‌ها و تجهیزات را در بر می‌گیرد. کارایی و بهره‌وری این سیستم‌ها به شکل قابل توجهی به کمک اتصالات نظیر به نظیر و مشتری-سرویس‌دهنده که محصول پیشرفت‌های جدید در اتصالات و کنترل کننده‌های ارزان قیمت و پروتکل‌های استاندارد اینترنت است، افزایش می‌یابد.

اینترنت اشیا، به اتصال فزاینده اشیا از هر نوع، از کاربردهای خانگی تا تجهیزات به کار رفته در کاربردهای صنعتی، به اینترنت یا ساختاری شبیه به اینترنت اشاره دارد. ایده‌ی اصلی در این مفهوم این است که تجهیزات هوشمند باید بتوانند با یکدیگر و با واسطهای انسانی در سراسر جهان برای افزایش بهره‌وری ارتباط برقرار کنند.

به عقیده‌ی برخی کارشناسان، IoT پس از اینترنت و شبکه‌های تلفن همراه، سومین موج از فناوری‌های ICT خواهد بود. انستیتو فرستر پیش‌بینی کرده است که ارتباطات اینترنتی شی با شی تا سال ۲۰۲۰ در حدود ۳۰ برابر تعاملات ارتباطی انسان با انسان خواهد بود و این بدان معناست که میلیاردها اتصال اینترنتی از آینده مخابرات، فقط مربوط به تعاملات بین اشیا خواهد بود.

انقلاب اینترنت منجر به ارتباط متقابل مردم با سرعت بی‌سابقه‌ای شده است. انقلاب آینده ارتباط میان اشیا برای ایجاد محیطی هوشمند خواهد بود. در سال ۲۰۱۱ تعداد دستگاه‌های متصل به یکدیگر به بیش از تعداد انسان‌های کره زمین رسید. بر اساس گزارش CISCO در سال ۲۰۱۲ حدود ۸,۷ میلیارد شی متصل به اینترنت جهانی وجود داشت. در سال ۲۰۱۳ این مقدار به بالغ بر ۱۰ میلیارد رسید. با کاهش هزینه برای هر اتصال و در نتیجه رشد سریع تعداد ارتباطات ماشین به ماشین، انتظار

می‌رود که این تعداد در سال ۲۰۲۰ به ۵۰ میلیارد برسد. پیش‌بینی می‌شود زمان مورد نیاز به منظور بلوغ اینترنت اشیا در صنعت بین ۵ تا ۱۰ سال است.

راه اندازی کارخانه‌های هوشمند سال‌های قبل نیازمند سیستم‌های اتوماسیون گران‌قیمت و سفارشی بود که تنها کارخانه‌های بزرگ از پس هزینه‌های آن برمی‌آمدند. شرایط در هوشمندسازی منازل نیز به همین ترتیب بود و تنها بیلیونرهایی مانند بیل گیتس از پس هزینه‌های سیستم‌های سفارشی هوشمندسازی منازل برمی‌آمدند. موارد زیر را می‌توان از عوامل گسترش اینترنت اشیا در سال‌های اخیر برشمرد:

- نصف شدن قیمت سنسورها و عملگرها در ۱۰ سال اخیر
- کوچکتر قدرتمندتر شدن سنسورها و عملگرها (سخت‌افزارها)
- دسترسی به ابزارات پشتیبانی مانند زیرساخت‌های ابری و ابزارات داده‌های عظیم

در اکو سیستم اینترنت اشیا، هوشمند بودن اشیا شرط لازم است ولی تنها فاکتور تعیین کننده نیست. اشیا هوشمند حجم بسیار بالایی از داده را از محیط حس می‌کنند. قدرت و ارزش اصلی راهکارهای مبتنی بر اینترنت اشیا در این نکته نهفته است که به چه نحوی بتوانند این داده‌ها را جمع‌آوری، تحلیل و براساس آن تصمیم‌سازی کنند. در معماری اینترنت اشیا، این مجموعه از وظایف بر عهده‌ی لایه پلتفرم است.

لایه‌ی پلتفرم به عنوان یک بستر مدیریت اشیا، جمع‌آوری داده‌ها، تحلیل آن‌ها و بستری برای توسعه برنامه‌های کاربردی است که بتوانند با دسترسی به این داده‌ها، تصمیمات لازم را اتخاد کرده و به اشیا ارسال کنند. در یک نگاه کلی، پلتفرم بی‌شباهت به سیستم عامل نیست چرا که همانند سیستم عامل می‌بایست جزئیات اشیا را از برنامه‌های کاربردی پنهان کرده و امکان دسترسی برنامه‌های کاربردی به اشیا را فراهم کند.

اما تفاوت مهمی بین سیستم عامل و پلتفرم وجود دارد. در اینترنت اشیا پلتفرم نه با یک سخت افزار بلکه با تعداد بسیار زیادی شی در تعامل است، بنابراین مقیاس‌پذیری آن از اهمیت بالایی برخوردار است. حسگرها داده کلان تولید می‌کنند که باید مدیریت شوند. برنامه‌های کاربردی هم می‌توانند از راه دور به پلتفرم متصل شوند بنابراین API مناسبی برای این منظور لازم است.

علیرغم نقش کلیدی پلتفرم در راه کارهای مبتنی بر اینترنت اشیا استاندارد مدونی در خصوص قابلیت‌ها و معماری این سامانه وجود ندارد. در این بین راهکارهای متعدد متن‌بازی ارائه شده‌اند که هر یک به نحوی به مسائل مقایس‌پذیری، خطاپذیری و ... پاسخ داده‌اند. هیچ یک از این راه‌کارها از معماری ریزسرویس‌ها استفاده نکرده‌اند و با توجه به پیچیدگی که در معماری آن‌ها وجود دارد نیاز به بستر قدرتمندی از دیدگاه سخت‌افزار برای اجرا دارند.

هدف از این پروژه، تولید پلتفرمی برای اینترنت اشیا با معماری ریزسرویس‌ها می‌باشد. در این پلتفرم نیازمندی‌های کلی برای یک پلتفرم عام منظوره، که در فصول آینده بیان شده است، در معماری ریز سرویس‌ها به صورت سرویس پیاده‌سازی می‌شوند. این سرویس‌ها در یک بستر مبتنی بر داکر[2] اجرا می‌شوند و وضعیت آن‌ها همانطور که در فصول آینده بیان شده است، مانیتور می‌گردد.

فصل دوم این پایان‌نامه به توضیح معماری ریزسرویس‌ها اختصاص دارد. فصل سوم به توصیف نیازمندی‌های پلتفرم عام منظوره اینترنت اشیا تخصیص یافته است. در فصل چهارم، درباره نحوه‌ی پیاده‌سازی پلتفرم و ارتباط آن با اشیا و برنامه‌های کاربردی بحث می‌شود. فصل پنجم به توضیح نحوه‌ی ارزیابی پلتفرم می‌پردازد. در نهایت، فصل ششم دربرگیرنده جمع‌بندی و کارهای پیشنهادی آینده پلتفرم خواهد بود.

[2] Docker

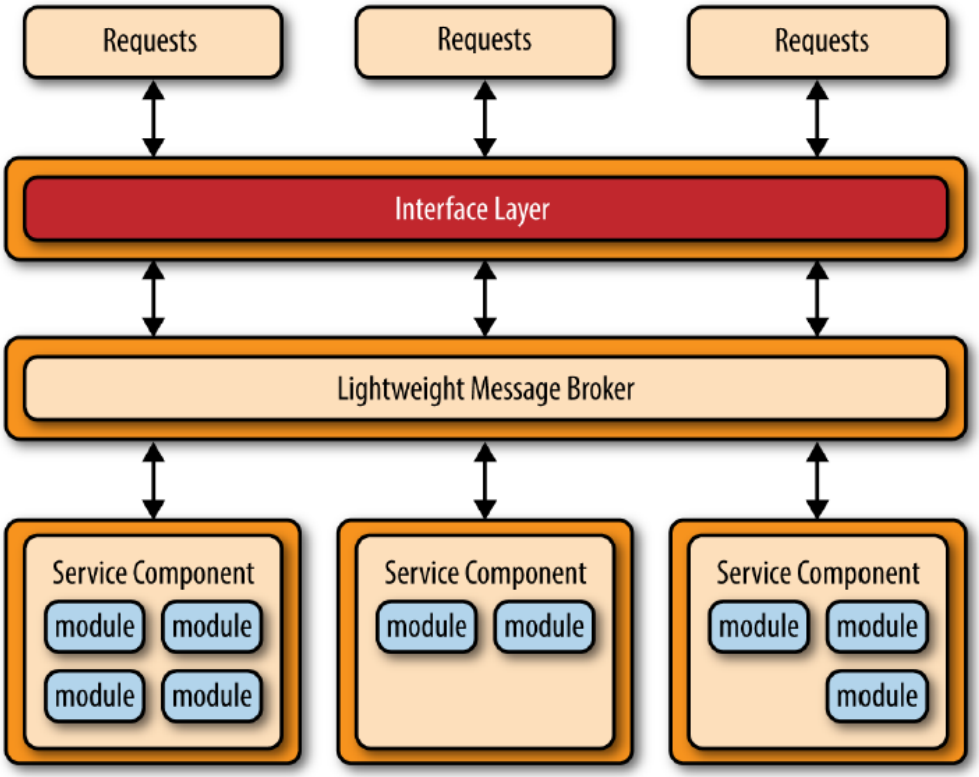
# ۲

# فصل دوم

# معماری ریزسرویس‌ها

# معماری ریزسرویس‌ها

ایده‌ی این معماری برخلاف روش‌های سنتی که سامانه‌هایی یکپارچه می‌باشند، یک سیستم توزیع شده متشکل از چندین سرویس است که هر سرویس به صورت مجزا پیاده‌سازی شده و سرویس مربوطه را از طریق API در اختیار سایر سرویس‌ها و مشتریان قرار می‌دهد. شمای کلی این معماری در شکل ۲-۱ نشان داده شده است.




**شکل ۲-۱ شمای کلی معماری ریزسرویس‌ها با توپولوژی centralized messaging [1]**

در این معماری، کارکردهای سامانه در قالب سرویس‌ها پیاده‌سازی می‌شوند. هر سرویس یک واحد قابل استقرار مستقل است، به این معنا که می‌تواند به تنهایی با نیازمندی‌های منحصر به فردش در یک سیستم اجرا شود. بستر اجرای این سرویس‌ها می‌تواند به شکل ماشین مجازی یا container باشد.

در حالی که راه‌های بسیار زیادی برای پیاده‌سازی معماری ریزسرویس‌ها وجود دارد، ۳ روش زیر در این بین شهرت بیشتری دارند:

1. API REST-Based Topology

2. Application REST-Based Topology
3. Centralized Messaging Topology

در توپولوژی centralized messaging برای توزیع درخواست‌های داده شده به سامانه و ارتباطات بین سرویس‌ها از یک مولفه به نام Message Broker استفاده می‌شود. این مولفه وظیفه‌ی توزیع پیام‌ها بین سرویس‌ها و بنابراین بخشی از وظیفه‌ی توزیع بار بین کامپوننت‌ها را بر عهده دارد.

با توجه به اینکه Message Broker یکی از اصلی‌ترین مولفه‌های این سیستم می‌باشد انتخاب آن باید به گونه‌ای باشد که بتواند به صورت توزیع شده عمل کند و مقیاس‌پذیر باشد زیرا در صورت از بین رفتن یا خرابی آن تمام سیستم دچار مشکل می‌گردد.

استفاده از این معماری مزایای مختلفی دارد. اکثر نیازمندی‌های سامانه می‌توانند به صورت سرویس‌های مجزا درقالب این معماری پیاده‌سازی شوند و هر یک از آن‌ها در صورت نیاز بارگذاری شوند. این معماری به دلیل توزیع‌شدگی به صورت ذاتی مقیاس‌پذیر می‌باشد و امکان اجرای همزمان چندین نمونه از یک سرویس برای توزیع بار را فراهم می‌آورد.

در کنار مزایای فراوان این معماری مشکلاتی هم دارد، توزیع سرویس‌ها به مولفه‌های مختلف باعث می‌شود داده‌های مشترک بین آن‌ها باز تعریف شوند و در صورت نیاز به تغییر این امر دشوار گردد و گاهی نیاز باشد کارکردهایی چندین بار پیاده‌سازی شوند. برای حل این مشکلات یکی از روش‌ها استفاده از زبان‌های برنامه‌نویسی است که ساخت و توزیع package ها در آن زبان آسان است و به این ترتیب می‌توان این کارکردها و داده‌های مشترک را به صورت package بین سرویس‌ها به اشتراک نهاد.

سایر توپولوژی‌هایی که برای پیاده‌سازی معماری ریزسرویس‌ها وجود دارند مبتنی بر پیاده‌سازی رابط برنامه‌نویسی REST بین سرویس‌ها می‌باشند این رابط که بر اساس HTTP عمل می‌کند سربار زیادی دارد چرا که پروتکل HTTP یک پروتکل متنی بوده و به این ترتیب سرآیند بسته‌های آن بسیار بزرگ می‌باشند و به همین علت استفاده از آن برای پیاده‌سازی وب‌سایت‌ها و ... که مستقیما با کاربران در ارتباط می‌باشند مناسب خواهد بود.

## ۲.۱ مولفه‌ی Message Broker

همانطور که پیشتر نیز اشاره شد، در معماری‌های مبتنی بر Message Broker این مولفه نقش حیاتی در سیستم ایفا می‌کند و سیستم ما نیز از این قاعده مستثنی نیست. برای پیاده‌سازی پلفترم به گونه‌ای که بتواند برآورده کننده نیازهای ما در گسترش‌پذیری باشد می‌بایست انتخابMessage Broker پیش از هر چیز صورت پذیرد. پیاده‌سازی‌های متن‌باز مختلفی از Message Broker ها موجود می‌باشد که هر یک از آن‌ها نیازمندی‌های مختلفی را برآورده می‌کنند در این بین چند مورد زیر شهرت بیشتری دارند.

### ۲.۱.۱ eMQ

این بروکر متن‌باز با استفاده از زبان Erlang نوشته شده است که یک زبان Functional بوده و قابلیت توزیع‌شدگی را به صورت درونی پشتیبانی می‌کند. پشتیانی از این بروکر از نسخه‌ی ۳,۱,۱ پروتکل MQTT به صورت کامل می‌باشد.

این بروکر توسط تست‌های صورت گرفته می‌تواند تا ۱۰ هزار پیام در ثانیه را با استفاده از سخت افزاری متوسط (۳۲ گیگابایت رم و ۸ پردازنده) مسیریابی[۳] کند. این بروکر می‌تواند با استفاده از پلاگین‌هایی که به زبان Erlang نوشته می‌شوند شخصی‌سازی شود.

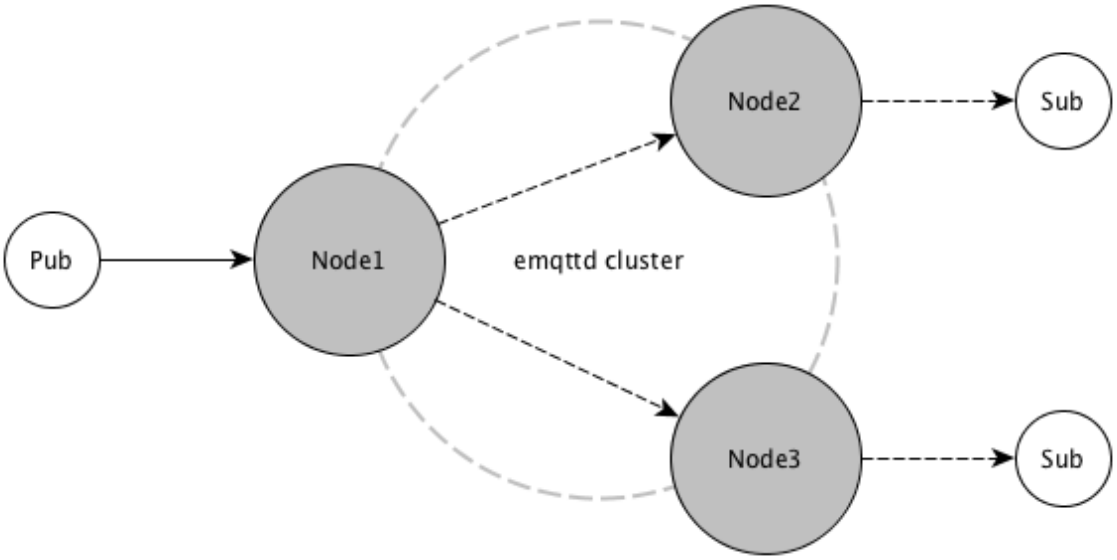
این بروکر می‌تواند به صورت توزیع شده و در یک کلاستر عمل کند. کلاسترینگ در emq مبتنی بر erlang/opt بوده و بر اساس دو قانون زیر شکل می‌گیرد:

۱. وقتی یک کلاینت mqtt بر روی یک موضوع مشترک می‌شود این امر به اطلاع تمامی گره‌های کلاستر می‌رسد.

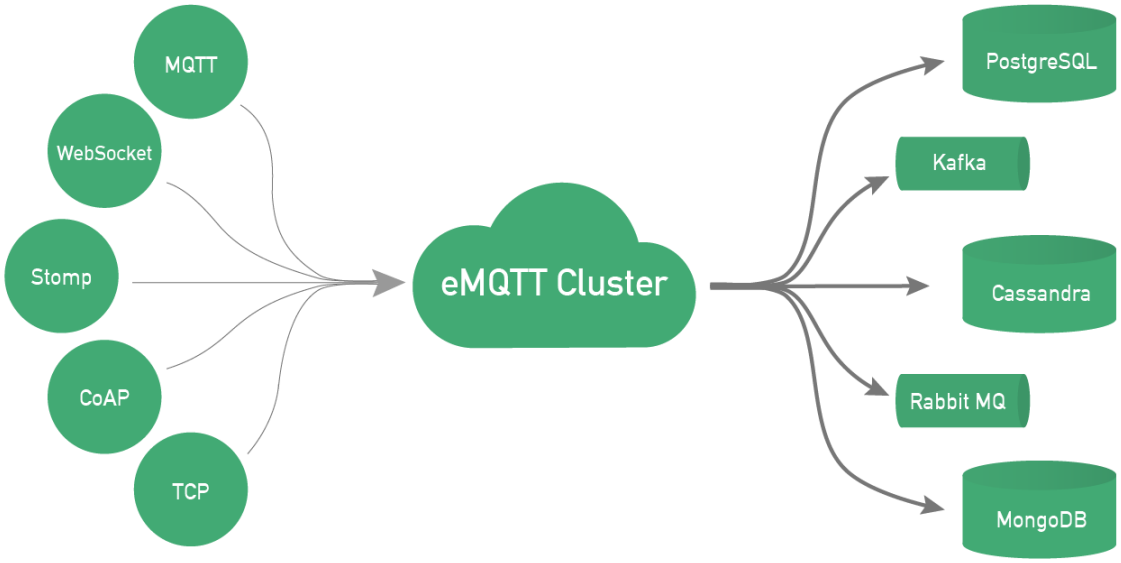
۲. وقتی پیامی توسط یک کلاینت mqtt منتشر می‌گردد، بر اساس یک جستجو در کلاستر این پیام به دست تمامی مشترکین آن در کلاستر می‌رسد.

---

[۳] در اینجا منظور از مسیریابی انتقال پیام از انتشار دهندگان به مشترکین یک موضوع می‌باشد.




**شکل ۲-۲ کلاسترینگ در emq**




**شکل ۲-۳ معماری بروکر eMQ**

### ۲.۱.۲ Aedes

Aedes در واقع یک بروکر نیست بلکه کتابخانه‌ای است که می‌توان با استفاده از آن یک بروکر را طراحی کرد. این کتابخانه به زبان NodeJs می‌باشد که زبانی است که به صورت ناهمگام طراحی شده و

کارآیی بالایی دارد. با توجه به اینکه تقسیم بار پلتفرم میان مولفه‌ها نیز قرار است در بروکر صورت بگیرد، استفاده از این کتابخانه قابلیت پیاده‌سازی این مهم و شخصی‌سازی‌های دیگر را در بروکر به ما می‌دهد. در این پروژه با استفاده از کتابخانه‌ی Aedes سرویس connectivity که در واقع یک بروکر و یک توزیع‌کننده‌ی بار است طراحی گشت.[2]

یکی از موضوعات مهم در سرویس connectivity کارآیی آن می‌باشد، جهت افزایش کارآیی این سرویس، از بحث clustering در زبان NodeJs استفاده شد، به این ترتیب که این سرویس از دو جز اصلی تشکیل master و worker. وظیفه‌ی master نگهداری لیست اشیا و سرویس‌های متصل است و تنها یک نمونه از آن می‌تواند در سیستم موجود باشد. وظیفه‌ی worker ارسال، دریافت و ترجمه پیام‌های mqtt می‌باشد. تعداد نمونه‌های موجود از workerها در سیستم می‌تواند یک یا بیشتر باشد و این تعداد توسط کاربر و در زمان اجرا قابل تظیم است.

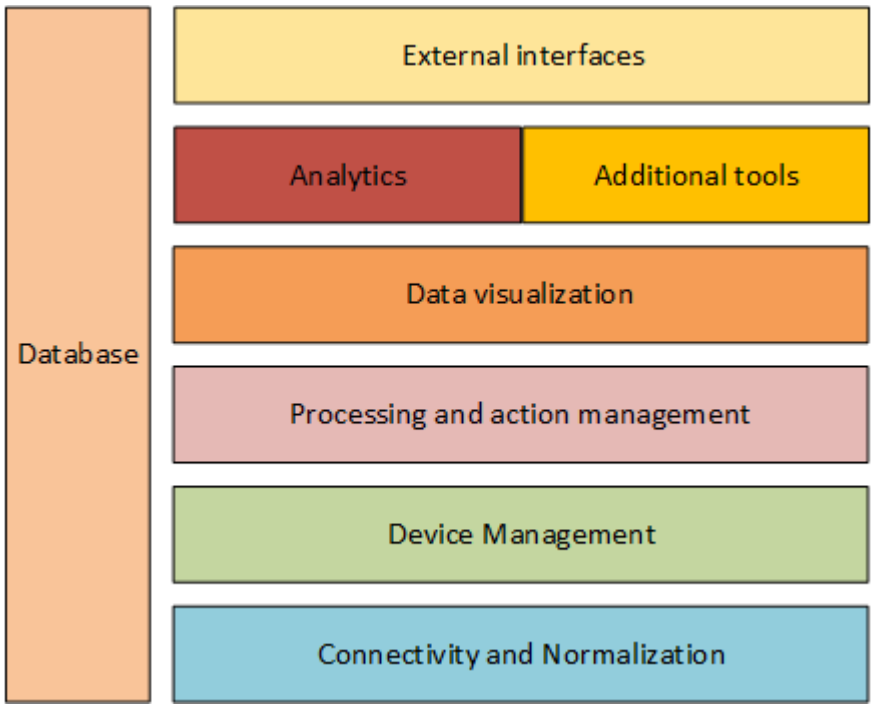
# ۳

# فصل سوم

# نیازمندی‌های پلتفرم عام منظوره

# نیازمندی‌های پلتفرم عام منظوره

یکی از بحث‌های اصلی در پیاده‌سازی یک پلتفرم عام منظوره برای اینترنت اشیا، پیدا کردن سرویس‌ها و ویژگی‌های اصلی یک پلتفرم می‌باشد. بررسی‌های مختلفی بر روی نیازمندی‌های یک پلتفرم عام منظوره برای اینترنت اشیا صورت گرفته است. یکی از این بررسی‌ها [2] می‌باشد که در آن سرویس‌های شکل ۳-۱ برای یک پلتفرم عام منظوره اینترنت اشیا لحاظ شده است.




**شکل ۳-۱ سرویس‌های پلتفرم عام منظوره اینترنت اشیا [2]**

۱. اتصال و نرمال‌سازی، در این قسمت اتصال‌های اشیا مدیریت می‌شوند و پیام‌های اشتباه و خطادار حذف می‌گردند.

۲. مدیریت دستگاه‌ها، در این بخش اشیا و دستگاه‌ها مدیریت می‌شوند و می‌توان سلامت آن‌ها را بررسی کرد. برای فراهم آوردن این سرویس نیاز است تا از سمت سخت افزار نیز پیاده‌سازی مدیریت خطا صورت گیرد.

۳. پردازش و مدیریت عمل‌ها، در این بخش می‌توان سناریوهایی از پیش تعیین شده را اجرا کرد یا سناریوهای جدید نوشت. سناریو در واقعا مجموعه‌ای از اعمال و شروط بوده که به صورت یک ماشین حالت متناهی[۴] می‌باشد، مثلا سناریو می‌تواند روشن شدن کولر در دمای ۳۰ درجه یا می‌تواند روشن چراغ در صورت ورود به اتاق باشد.

۴. قسمت‌های نمایش داده و آنالیز آن بیشتر جنبه‌ی داده‌کاوی دارند و نیازمندی‌های اولیه را برای نرم افزارهای هوشمندی که بر روی پلتفرم نوشته می‌شوند، فراهم می‌آورند. عموما بستر های موجود در این قسمت فعالیت خاصی نکرده‌اند و بیشتر تونل‌هایی برای ارسال streamهای داده فراهم آورده‌اند. نرم افزارهای متن باز مختلفی وجود دارند که جهت پردازش داده می‌توان از آن‌ها استفاده کرد و فراهم کردن نیازمندی‌های اولیه برای همه‌ی آن‌ها در یک بستر ممکن نیست، به همین روی عموما بسترها به تعدادی از این قبیل نرم افزارها که معروفتر هستند متصل می‌شوند.

در ادامه به بررسی سرویس‌هایی می‌پردازیم که قرار است در بستر ما پیاده‌سازی شوند. این سرویس‌ها در نهایت و در کنار یکدیگر آنچه در [2] آمده است را شکل می‌دهند. در عمل هر یک از این سرویس‌ها به صورت یک ریزسرویس پیاده‌سازی می‌گردند.

## ۳.۱ سرویس Log

سخت افزارها در اینترنت اشیا می‌بایست به گونه‌ای طراحی شوند که مصرف انرژی در آن‌ها حداقل باشد، یکی از این روش‌ها به این صورت است که سخت افزار به خواب می‌رود و به صورت متناوب از خواب بیدار شده و داده‌ای را ارسال می‌کند.

[4] Finite State Machine (FSM)

پلفترم‌ها می‌بایست یک سرویس مشخص برای جمع‌آوری چنین داده‌هایی داشته باشند، ما به صورت غیر رسمی با توجه به آنچه در پلتفرم Kaa [3] معرفی شده است این سرویس را سرویس Log می‌نامیم.

**جدول ۳-۱ نیازمندی‌های سرویس Log**

| ردیف | کد | نیازمندی | ملاحضات |
|---|---|---|---|
| ۱ | Log-1 | دریافت و ذخیره‌سازی در پایگاه داده، داده‌هایی که به صورت پریودیک توسط سنسورها ارسال می‌گردند. | |
| ۲ | Log-2 | پاسخ به درخواست‌های کاربر برای دریافت داده‌هایی که در پایگاه داده ذخیره شده‌اند. | |
| ۳ | Log-3 | پشتیبانی از چندین پایگاه داده‌ای متفاوت | |

## ۳.۲ سرویس Configuration یا Set

در هر محیط هوشمندی سخت افزارهایی با نام عملگر وجود دارند و با توجه به تنظیماتی که دارند می‌توانند پارامترهایی را در محیط کنترل کنند، وظیفه‌ی پلتفرم اینترنت اشیا می‌باشد که این تنظیمات را در اختیار برنامه‌های هوشمند قرار دهد.

وظیفه‌ی سرویس set کنترل تنظیمات عملگرها می‌باشد. این سرویس می‌تواند به دو صورت پیاده‌سازی شود، در روش اول سرویس set به ازای پیام‌هایی که ارسال می‌کند انتظار جوابی را نداشته و در این صورت تقاضا دهنده نمی‌تواند متوجه شود تنظیماتی که خواسته است اعمال شده‌اند یا خیر، این چنین پیاده‌سازی نیاز به سخت افزار خاصی ندارد و می‌تواند با ساده‌ترین سخت افزارها کار کند، در روش دوم سرویس set به ازای پیام‌هایی که ارسال می‌کند انتظار جواب داشته و به این ترتیب تقاضا دهنده در جریان وضعیت تنظیماتی که اعمال کرده است قرار می‌گیرد، این روش نیاز به سخت افزار پیچیده‌تری نسبت به روش اول دارد و این بستر ما از روش دوم استفاده می‌کنیم.

**جدول ۳-۲ نیازمندی‌های سرویس Set**

| ردیف | کد | نیازمندی | ملاحضات |
|---|---|---|---|
| ۱ | Set-1 | تغییر تنظیمات روی عملگرها با توجه به درخواست کاربر، پیگیری وضعیت درخواست و به روزرسانی پایگاه داده‌ی تنظیمات | |
| ۲ | Set-2 | پاسخ به درخواست‌های کاربر برای دریافت آخرین تنظیمات عملگرها | |
| ۳ | Set-3 | پشتیبانی از انجام چندین درخواست به صورت همروند | |

## ۳.۳ سرویس Event

در سرویس Log به چگونگی جمع آوری داده‌های یک سنسور در زمانی که آن‌ها را به صورت پریودیک ارسال می‌کند پرداختیم، در این میان اگر فرض کنیم داده‌هایی وجود دارند که از جنس رویداد هستند، مانند تشخیص حضور[5] این داده‌ها ارزش‌های لحظه‌ای دارند و می‌بایست در لحظه به کاربر اطلاع داده شوند، از این رو سرویس Event با فراهم آوردن یک سرویس بهنگام[6] کاربر را از این گونه رویدادها مطلع می‌سازد.

[5] Motion Detection

[6] Real-time

**جدول ۳-۳ نیازمندی‌های سرویس Event**

| ردیف | کد | نیازمندی | ملاحضات |
|---|---|---|---|
| ۱ | Event-1 | فراهم آوردن بستر ارتباط بهنگام برای کاربر | |
| ۲ | Event-2 | ارسال داده‌های رویدادهایی که از سنسورها دریافت می‌شود به کاربر | |

## ۳.۴ سرویس Get

داده‌های مختلفی روی هر عملگر یا سنسور وجود دارند که با سرویس‌های Log یا Event نمی‌توان آن‌ها را جمع‌آوری کرد. این دادگان شامل نام سازنده‌ی سنسور، زمان ساخت و ... می‌باشند. برای جمع‌آوری دادگانی از این دست کاربر تقاضا می‌دهد و بستر داده‌ی مورد نظر را از شی گرفته و به کاربر باز می‌فرستد.

**جدول ۳-۴ نیازمندی‌های سرویس Get**

| ردیف | کد | نیازمندی | ملاحضات |
|---|---|---|---|
| ۱ | Get-1 | دریافت داده‌های مورد تقاضای کاربر، از اشیا و باز ارسال آن‌ها به کاربر | |
| ۲ | Get-2 | پشتیبانی از انجام چندین درخواست به صورت همروند | |

## ۳.۵ سرویس مدیریت دستگاه‌ها

در یک بستر اینترنت اشیا نیاز است که آماری دقیق از دستگاه‌ها و انواع آن‌ها در دست باشد، این امر تنها محدود به اینترنت اشیا نبوده و در مدیریت شبکه نیز مطرح بوده است و راه حل‌های مختلفی برای آن طراحی و پیشنهاد شده است. یکی از این راه‌ها که از مدیریت شبکه می‌توان در اینجا استفاده کرد، طراحی یک زبان و توصیف اشیا و ویژگی‌های آن‌ها با این زبان است.

## ۳.۶ نگاشت سرویس‌ها به نیازمندی‌های پروپوزال

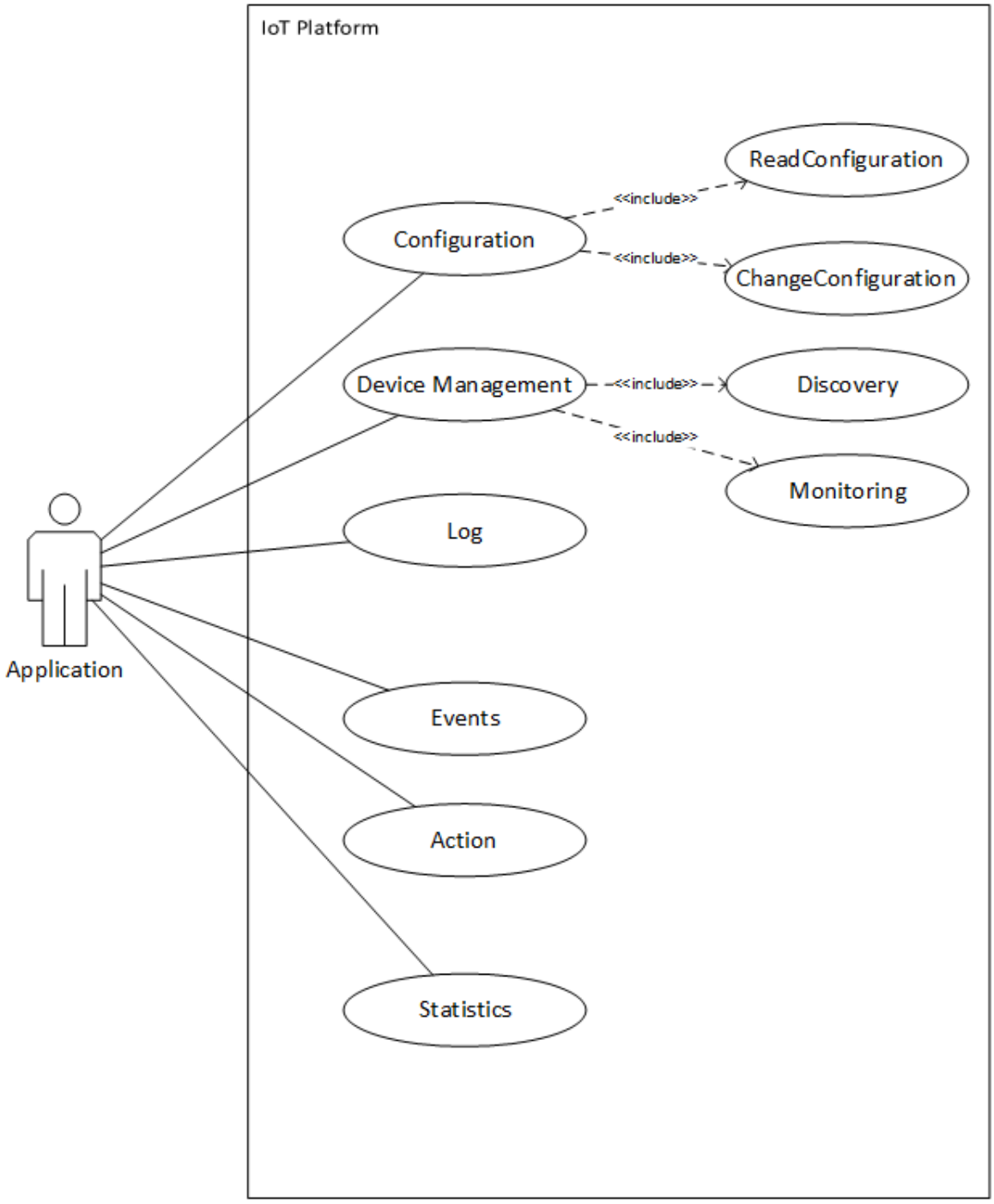
در ادامه به نگاشت سرویس‌هایی که به اینجا معرفی کرده‌ایم به نیازمندی‌هایی که در پروپوزال پروژه آمده است می‌پردازیم. همانطور که در پروپوزال هم بیان شده بود سیستم از دو دیدگاه قابل بررسی هست، دیدگاه اول دیدگاه برنامه‌هایی هستند که بر روی بستر ما اجرا می‌شوند و دیدگاه دوم دیدگاه سخت افزارهایی است که به بستر ما متصل می‌شوند. هر یک از این دیدگاه‌ها نیازمندی‌های خود را دارند و نمی‌توان آن‌ها را به صورت مستقل از هم تصور کرد، به همین دلیل برای هر یک از آن‌ها یک use-case ارائه کردیم. در واقع آنچه در use-case سمت سخت افزار قرار است که فراهم شود می‌بایست سرویس‌هایی باشد که قرار است به برنامه‌های کاربردی ارائه شود به این معنی که نمی‌توان سرویس Log را به کاربر ارائه داد زمانی که در سمت سخت افزار توانایی جمع‌آوری آن وجود ندارد. سرویس‌هایی که در قسمت قبل معرفی شدند مطابق با جدول زیر به نیازمندی‌هایی که در ادامه در use-caseها آمده‌اند نگاشت می‌گردند، توجه به این نکته خالی از لطف نیست که هر سرویس دو قسمت شمالی و جنوبی دارد که هر یک به ترتیب به برنامه‌های کاربردی و سخت‌افزارها سرویس می‌دهند.

**جدول ۳-۵ نگاشت سرویس‌ها و نیازمندی‌ها**

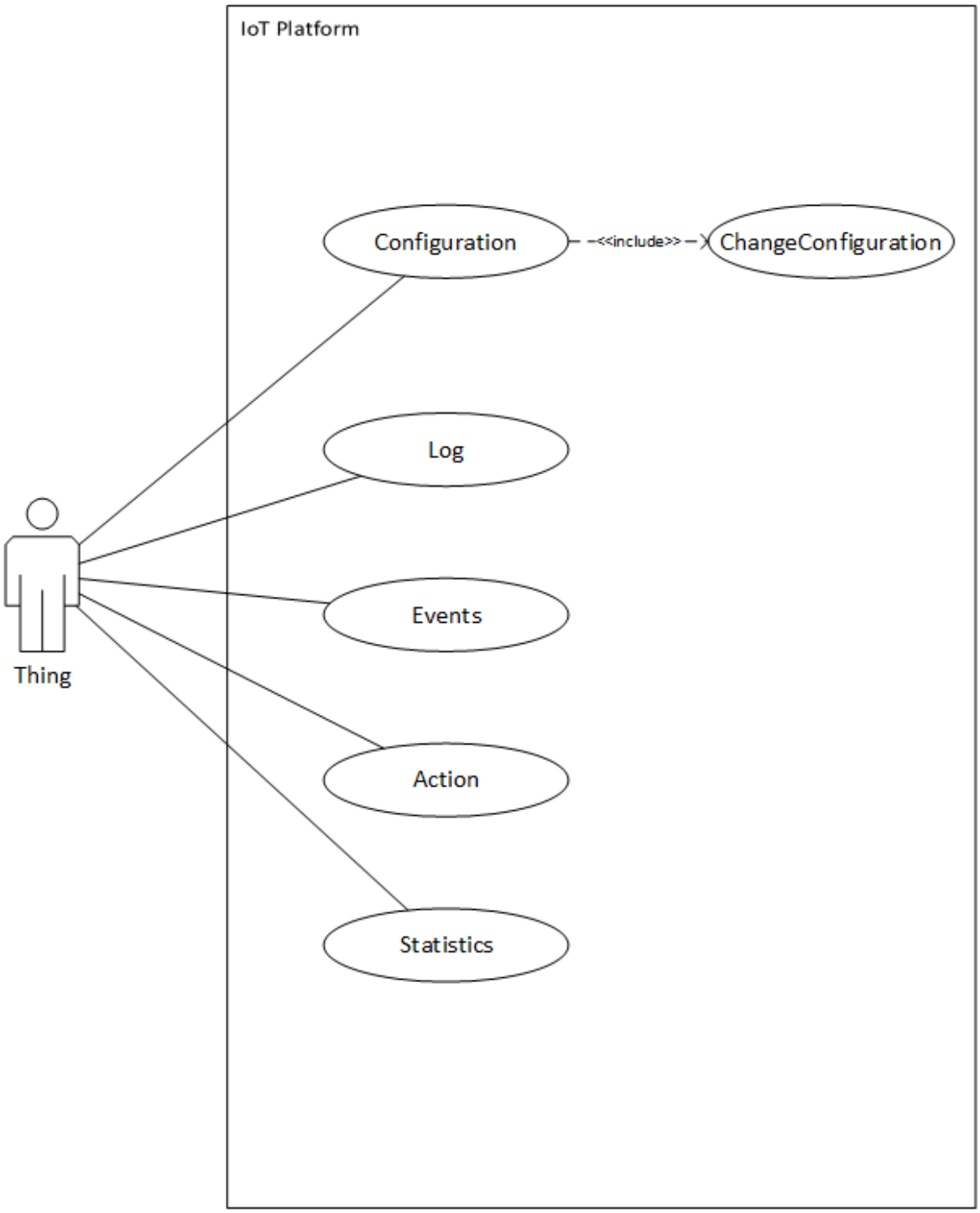
| نیازمندی | سرویس |
|---|---|
| Configuration | Set |
| Device Management | مدیریت دستگاه‌ها |
| Events | Events |
| Statistics | Get |
| Log | Log |
| Action | - |

همانطور که جدول نیز آمده است نیازمندی action به علت آنکه بخش عمده‌ی آن از طریق سرویس Set قابل پیاده‌سازی است، پیاده‌سازی نشده است، از سوی دیگر پیاده‌سازی نیازمندی‌های

سخت افزاری آن به گونه‌ای است که عملا پیاده‌سازی آن بار اضافی بر سیستم اعمال می‌کند.




**شکل ۳-۲ نمودار use-case از دیدگاه application**




**شکل ۳-۳ نمودار use-case از دیدگاه اشیا**

## ۳.۷ نیازمندی‌های غیرکارکردی

یکی از جنبه‌های هر سیستم نرم‌افزاری نیازمندی‌های غیر کارکردی آن می‌باشد، این نیازمندی‌ها می‌توانند بسیار متنوع باشند. در محیط‌های مبتنی بر اینترنت اشیا امنیت و گسترش‌پذیری نقش به سزایی را بر عهده دارند، چرا که افزایش تعداد اشیا در یک محیط مبتنی بر اینترنت اشیا چیزی دور از انتظار نبوده و امنیت نیز یکی از دغدغه‌هایی است که به علت گسترده بودن سطح نفوذ در اینترنت اشیا بسیار حائز اهمیت است. گسترش‌پذیری، کارآیی و دسترس‌پذیری در پلتفرم را در فصل ۵ به تفضیل بررسی خواهیم کرد. در این قسمت نگاهی به نیازمندی‌های امنیتی در یک پلتفرم از دیدگاه [4] می‌اندازیم.

۱. شناخت: در یک محیط مبتنی بر اینترنت اشیا می‌بایست تمامی بازیکنان مانند سازندگان حسگرها و عملگرها، فراهم کنندگان بستر ارتباطی و ... را شناخت و مطمئن شد که سطحی از امنیت که مورد نظر ما است را فراهم کرده‌اند.

۲. امینت به صورت انتها به انتها: تک تک لایه‌های یک محیط مبتنی بر اینترنت اشیا می‌بایست از نظر امینتی در رابطه با احراز شناسایی[۷] رمزنگاری[۸] و احراز هویت[۹] بررسی شوند به گونه‌ای که امنیت پلتفرم و کاربر را تضمین کنند.

۳. ارتباطات: انتخاب بستر ارتباطی با اشیا باید به گونه‌ای باشد که نیازهای امنیتی ما را برآورده سازد. پروتکل‌های ارتباطی مختلفی برای انتقال داده با اشیا وجود دارند مانند: پروتکل Zigbee، پروتکل IEEE802.11 (Wi-Fi) و پروتکل‌های Cellular و ... هر یک از این پروتکل‌ها مزایا و معایب خود را دارند و می‌توانند پشتیانی‌های امنیتی خاص خود را ارائه دهند.

[7] Identification

[8] Encryption

[9] Authentication

۴. پیروی از استانداردها: آمادگی برای پیروی از قوانین مربوط به جمع‌آوری، نگهداری و استفاده از داده‌های شخصی یا مالی کاربر می‌بایست همواره وجود داشته باشد چرا که با توجه به رشد روز افزون اینترنت اشیا این قوانین ممکن است به زودی در سطح ملی یا جهانی به تصویب برسند.

پلتفرم اینترنت اشیا به عنوان جزئی از اکوسیستم اینترنت اشیا با توجه به آنچه پیشتر بیان شد باید بتواند نیازمندی‌های امنیتی شناسایی، رمزنگاری و احراز هویت را انجام دهد. در پلتفرم توسعه‌یافته توسط ما این نیازمندی‌های پیاده‌سازی نشدند، ولی امکان پیاده‌سازی آن‌ها توسط کاربر فراهم است و کاربر می‌تواند در صورت نیاز با استفاده از TLS و DTLS برای فراهم آوردن رمزنگاری بر روی پروتکل‌های ارتباطی پلتفرم اقدام کند.

یکی دیگر از نیازهای امنیتی در پلتفرم احراز هویت اشیا متصل شده می‌باشد، در این پلتفرم هر agent در زمان اتصال یک عدد به عنوان شناسه دریافت می‌کند و تمامی پیام‌های بعد از آن به وسیله‌ی این شناسه ارسال می‌گردند، این شناسه به صورت یکتا بر اساس نامی که برای agent انتخاب شده است تولید می‌شود و نمی‌توان با جعل نام شی وارد سیستم شد.

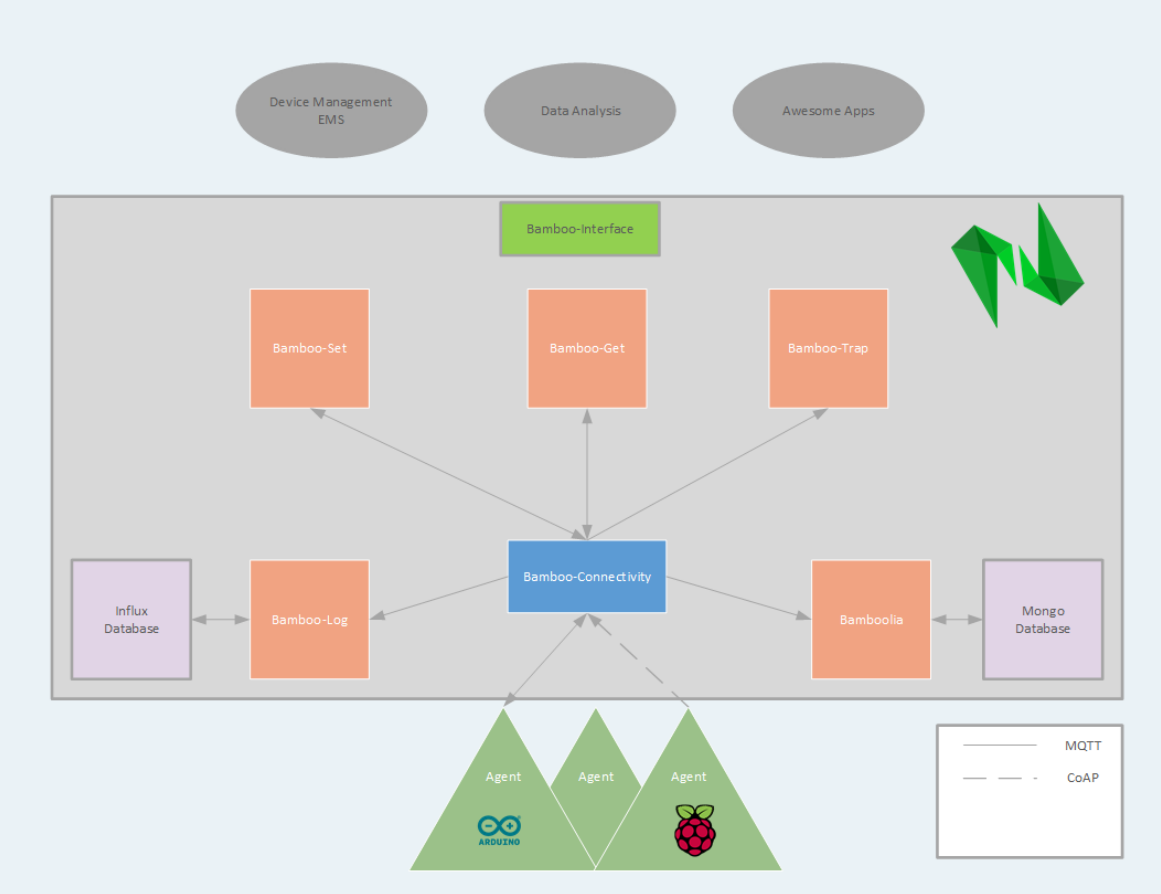
# ۴

# فصل چهارم

# پیاده‌سازی

# پیاده‌سازی

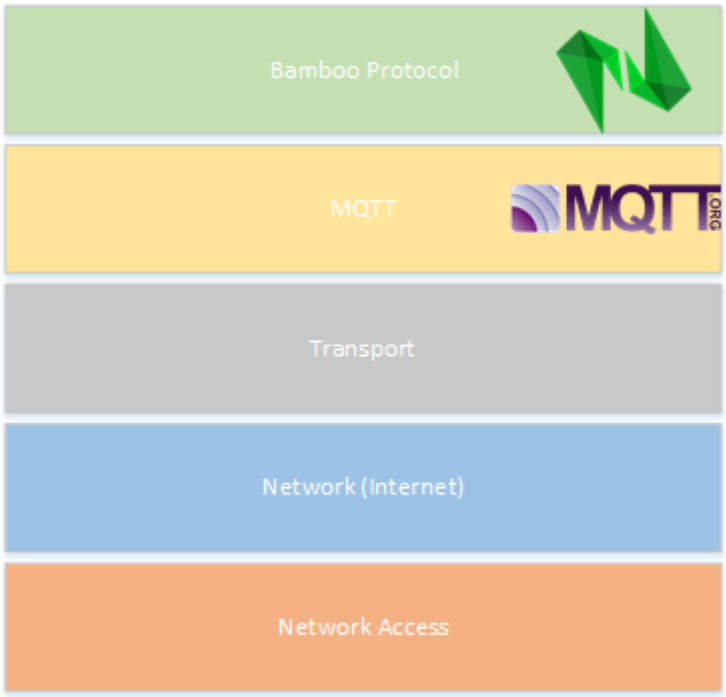
پلتفرم حاصل از این پروژه بامبو نام دارد، دلیل این نامگذاری از این بابت است که این پلتفرم در صورت از کار افتادن هر یک از سرویس‌های آن می‌تواند دوباره به کار خود ادامه دهد و سرویس‌های از کار افتاده را احیا کند. همانطور که در فصل‌های پیشین نیز شرح داده شد برای پیاده‌سازی این پلتفرم از معماری ریزسرویس‌ها و زبان NodeJs استفاده شد. در روند پیاده‌سازی سرویس‌ها با توجه به نیازمندی‌هایی که فصل پیشین شرح داده شد پیاده‌سازی شدند. معماری کلی پلتفرم به شکل ۴-۱ است:




**شکل ۴-۱ معماری کلی پلتفرم بامبو**

اشیا (شامل سنسورها و عملگرها) می‌توانند به صورت مستقیم یا غیر مستقیم به پلتفرم متصل شوند، بنابر تعریف agent هر آن چیزی است که می‌تواند پشته‌ی پروتکلی لازم جهت ارتباط با پلتفرم را پیاده‌سازی کند. بنابراین اشیایی که به صورت مستقیم به پلتفرم متصل می‌شوند گویی agent نیز

می‌باشند و اشیایی که نمی‌توانند به صورت مستقیم به پلتفرم متصل شوند جهت اتصال نیاز به یک agent خواهند داشت.

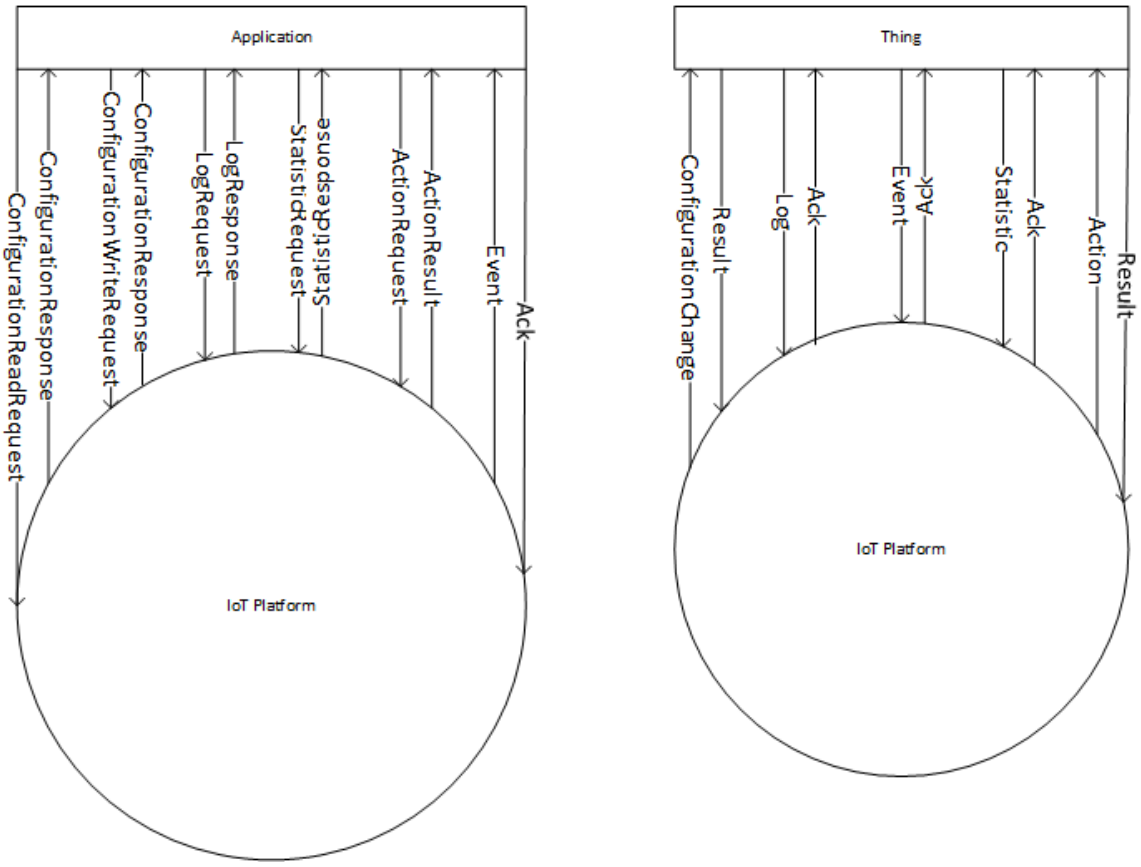



**شکل ۴-۲ پشته‌ی پروتکلی پلتفرم بامبو**

همانطور که در شکل ۴-۲ مشهود است پشته‌ی پروتکلی پلتفرم جهت ارتباط با agentها از mqtt و TCP/IP استفاده می‌کند. mqtt یک پروتکل سبک جهت انتشار و مشترک شدن است که در کاربردهای مختلفی از جمله اینترنت اشیا استفاده می‌گردد. [4] در اینترنت اشیا پشته‌های پروتکلی گوناگونی وجود دارند که بعضی از آن‌ها از mqtt نیز پشتیبانی می‌کنند ولی به دلیل آنکه هنوز محبوبیت چشم‌گیری پیدا نکرده‌اند و نمی‌توان از آن‌ها در شبکه‌های WAN استفاده کرد در این پلتفرم تصمیم گرفتیم تا از TCP/IP استفاده کرده و به این ترتیب اجازه دهیم پلتفرم و agentها از طریق شبکه WAN نیز در ارتباط باشند.

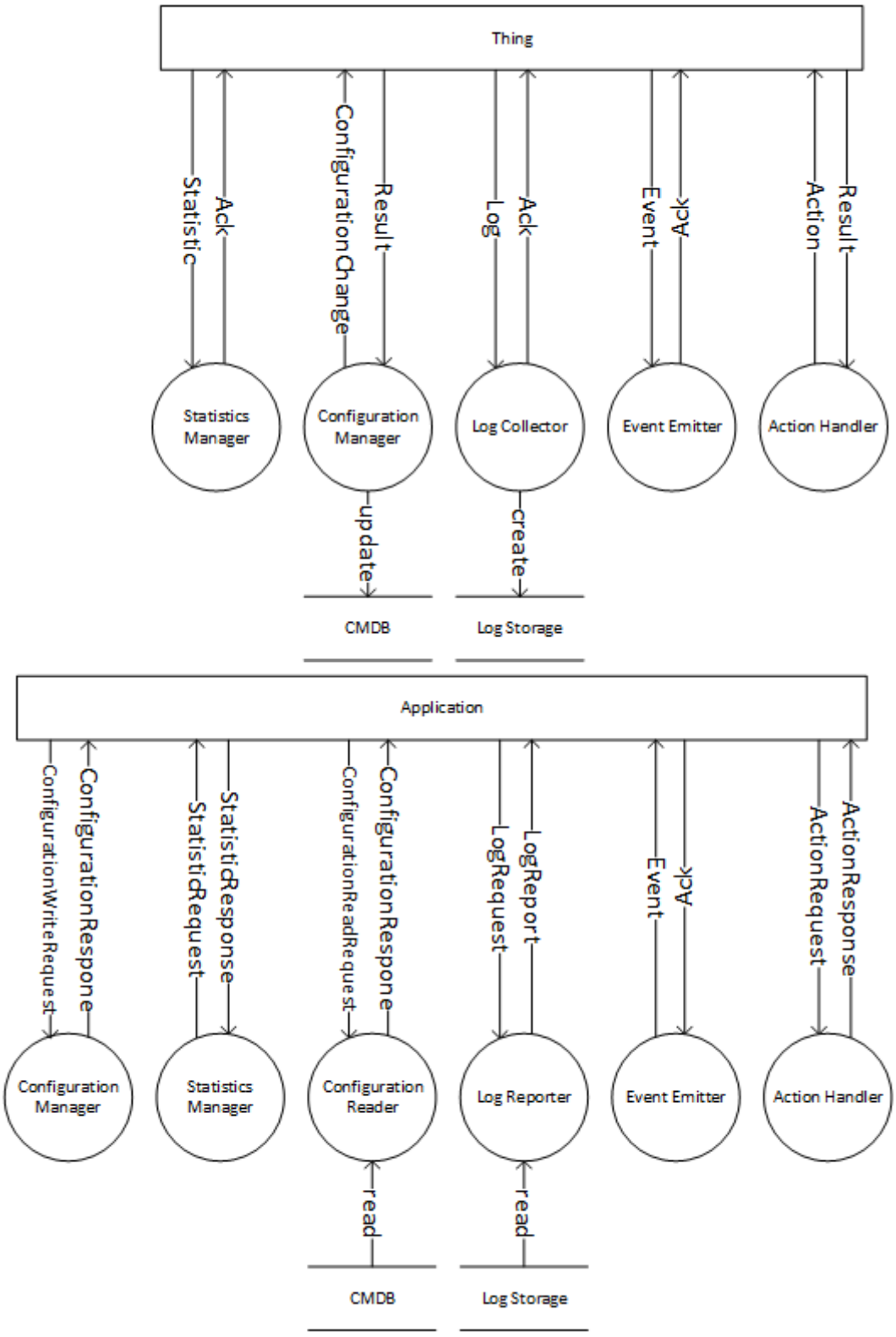
ذکر این نکته خالی از لطف نیست که استفاده از agentها این قابلیت را فراهم می‌آورد که اشیا خود در یک شبکه WSN[1] قرار داشته باشند و داده‌های نهایی از طریق agent برای پلتفرم ارسال کنند، به این ترتیب در صورت نیاز به پشتیبانی از محدودیت‌هایی خاص مانند محدودیت توان مصرفی اشیا می‌توان این امر را با استفاده از پروتکل‌هایی از WSN که برای مصارف خاص طراحی شده‌اند، برآورده ساخت.

همانطور که در پروپوزال نیز بیان شده بود، ارتباطات شی-پلتفرم و کاربر-پلتفرم را می‌توان در نمودارهای جریان داده‌ای شکل ۴-۳ و شکل ۴-۴ خلاصه کرد.



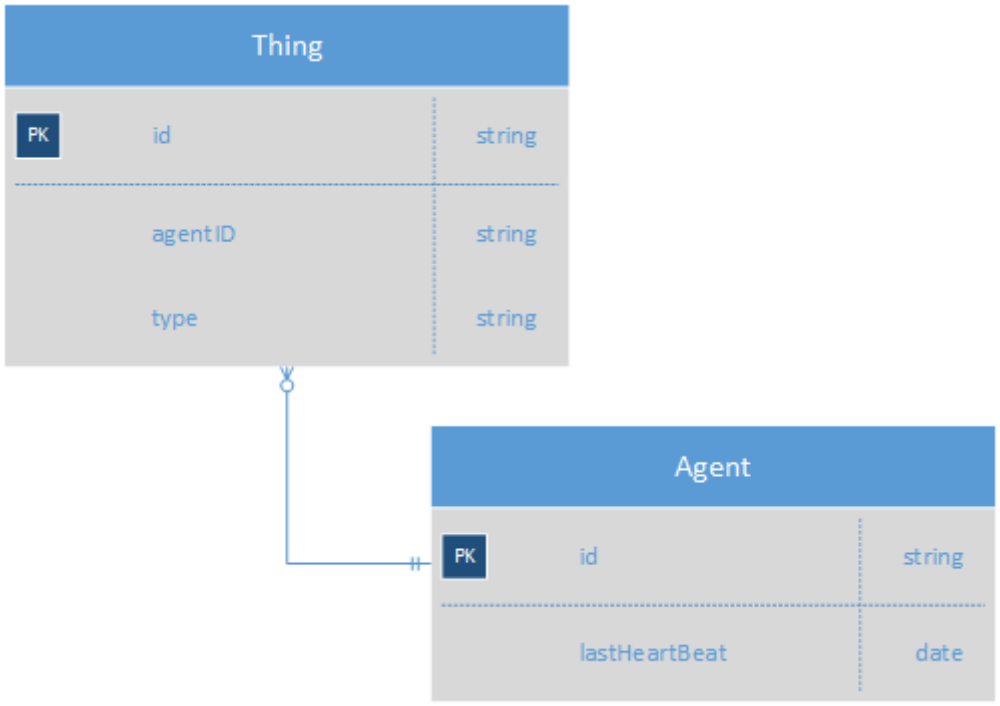

**شکل ۴-۳ نمودار جریان داده‌ای سطح ۰**

[1] Wireless Sensor Network




**شکل ۴-۴ نمودار جریان داده‌ای سطح ۱**

برای نگهداری ارتباط اشیا و agentها یک از یک پایگاه داده‌ای NoSQL استفاده شد، زیرا این روبط بسیار ناپایدار بوده و می‌توانند به سرعت در اثر خرابی هر یک از اجزای سیستم (اشیا یا agentها) تغییر کنند، نمودار ERD مربوط به اطلاعات اشیا و agentها در شکل ۴-۵ آمده است:



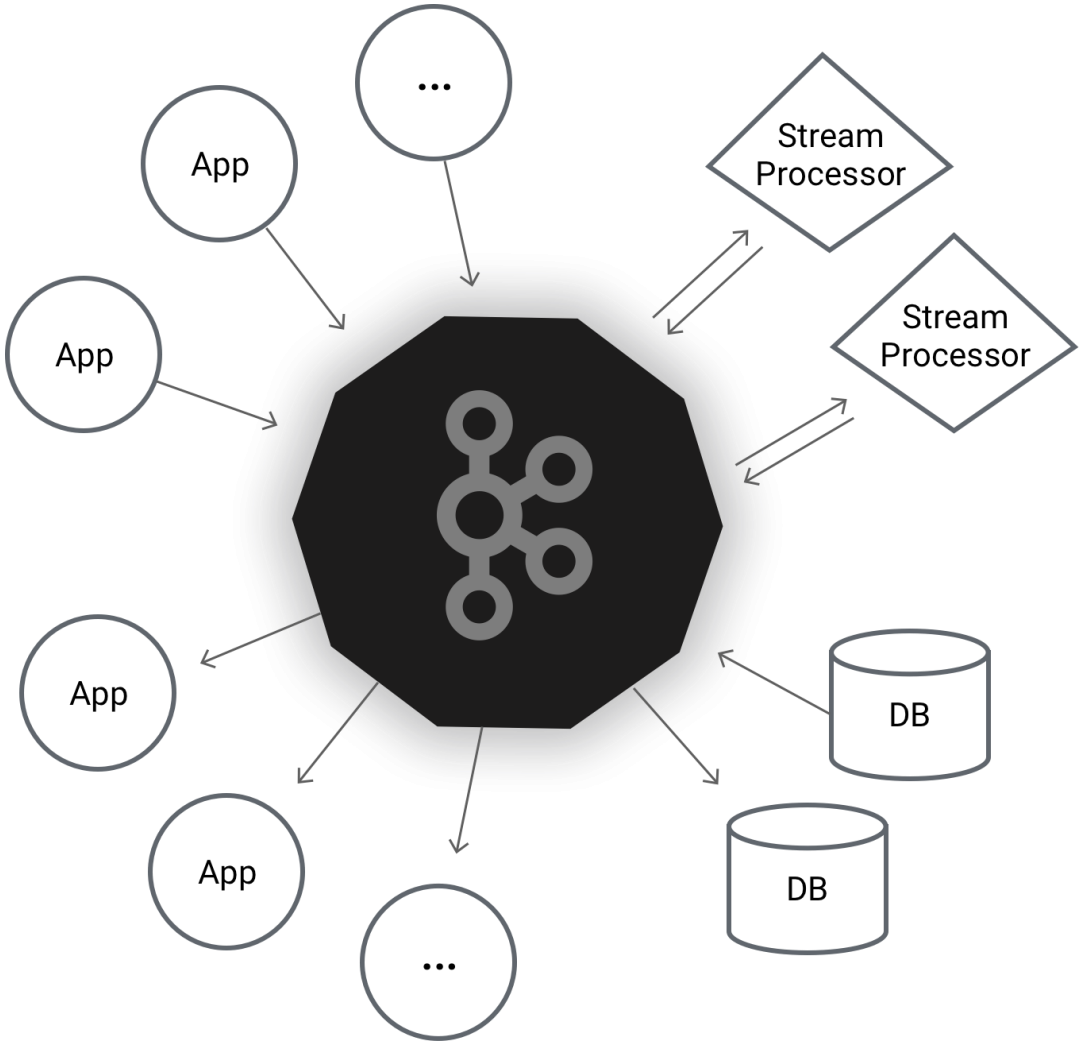

**شکل ۴-۵ نمودار ERD مربوط به اشیا و agentها**

همانطور که در این معماری نیز مشهود است پروژه از سرویس‌هایی که پیشتر معرفی شده‌اند تشکیل شده که هر سرویس در قالب یک مولفه پیاده‌سازی شده است. برای اینکه از پیاده‌سازی چندباره‌ی قسمت‌های مشترک بین این مولفه‌ها پرهیز گردد، کتابخانه‌ای تحت عنوان کامپوننت پیاده‌سازی گشت که در آن نیازمندی‌های مشترک بین componentها دیده شده است. [5]

برای سرویس Log یک کامپوننت به اسم Log پیاده‌سازی شده است که در آن از پایگاه داده‌ای Influx استفاده شده است، این پایگاه داده به گونه‌ای طراحی شده است که برای نگهداری داده‌هایی که در زمان تولید می‌گردند بهینه باشد. این کامپوننت به گونه‌ای طراحی شد که بتوان با پایگاه‌های داده‌ای دیگر نیز با آن کار کرد. [6]

یکی از دلایلی که در رابطه با وجود پلتفرم در اکوسیستم اشیا پیشتر بیان شد، فراهم آوردن امکان تحلیل داده‌ای و ... بود. ابزارهای بسیاری جهت تحلیل داده موجود می‌باشند که از جمله‌ی آن‌ها می‌توان

به Apache Spark اشاره کرد. در پیاده‌سازی انجام شده از پلتفرم توسط ما سرویس Log می‌تواند به عنوان پایگاه داده‌ای، از Apache Kafka استفاده کند. Apache Kafka در واقع یک بستر جریانی[11] می‌باشد که می‌تواند جریان داده‌ای که به آن وارد می‌شود را با فرمت مناسب به تعدادی از پردازنده‌های داده‌ای مشهور از جمله Apache Kafka خروجی دهد. البته Apache Kafka ویژگی‌های بسیار بیشتری نیز دارد که از حوصله‌ی این بحث خارج است.




**شکل ۴-۶ Apache Kafka در یک نگاه**

[1] Streaming Platform [1]

برای سرویس Trap یک مولفه با نام Trap پیاده‌سازی شده است که در آن با استفاده از کتابخانه‌ی Socket.io ارتباط همزمان برقرار می‌گردد. ویژگی برجسته‌ی socket.io [7] از این حیث است که تاخیر ارتباط در آن بسیار اندک بوده (در حد چند میلی‌ثانیه برای ارتباطات محلی) و می‌تواند تعداد بسیار زیادی از ارتباطات را پشتیبانی کند. [8] جهت اطمینان از کارکرد و کارآیی socket.io تعدادی تست مستقل از پلتفرم از این بستر در ارتباط‌های LAN و WAN گرفته شد. [11]

برای سرویس‌های Get و Set از آنجایی که پشتیانی سخت افزاری برای ارسال پاسخ از سوی سخت افزار به پلتفرم وجود نداشت، پیاده‌سازی بسیار محدودتری صورت گرفت، به این ترتیب که سرویس Get پیاده‌سازی نشد و در سرویس Set قسمت پاسخ در ازای ارسال یک عمل دیده نشد، به این ترتیب در ازای تنظیم شدن یک پارامتر کاربر پاسخی مبنی بر وضعیت نهایی دریافت نخواهد کرد. [12]

ارتباط کاربر و برنامه‌های کاربردی با پلتفرم از دوطریق رابط کاربری و رابط برنامه نویسی (API) صورت می‌پذیرد. هر سرویس از یک رابط برنامه نویسی مستقل که مبتنی بر REST می‌باشد بهره می‌برد. این رابط‌های کاربری در مولفه‌ی API مجتمع می‌شوند و این امکان را برای کاربر فراهم می‌آورند که تنها با یک نقطه ارتباط برقرار کرده و توزیع‌شدگی سیستم را از دید او پنهان می‌کنند. این مولفه ضمن تشخیص سرویس مورد نظر کاربر از تقاضای او، تقاضای او را به صورت تصادفی میان سرویس‌ها توزیع می‌کند و به این ترتیب بار سیستم را کاهش می‌دهد. رابط کاربری پلتفرم بسیار ساده بوده و مبتنی بر فریم ورک Nuxt و Vue.js می‌باشد.

مدل‌ها و وضعیت اشیا در مولفه‌ای تحت نام Bamboolia مدیریت می‌شوند، این مولفه در واقع فراهم آورنده‌ی یک زبان برای توصیف اشیا نیز است، که به وسیله‌ی آن می‌توان اشیا را توصیف کرده و مدل اطلاعاتی آن را مشخص کرد. جهت جمع‌آوری اطلاعات اشیا متصل به پلفترم و .. این سرویس با سرویس connectivity در ارتباط است و اتصال و عدم اتصال اشیا را از طریق آن سرویس پیگیری می‌کند.[13]

در ادامه نمونه‌ای از توصیف مدل اطلاعاتی یک لامپ در زبان مذکور آورده شده است:

```
---
name: lamp
package: .standard

attributes:
  - name: vendor
    type: string
settings:
  - name: on
    type: boolean
```

سرویس‌ها به صورت کانتینر[۱۲]اجرا و مدیریت می‌شوند، کانتینرها به مانند ماشین‌های مجازی[۱۳] رفتار می‌کنند، با این تفاوت که سربار ساخت و باز تولید آن‌ها بسیار کم و می‌توان به سادگی تعداد زیادی از آن‌ها را تولید و در صورت لزوم باز تولید نمود.

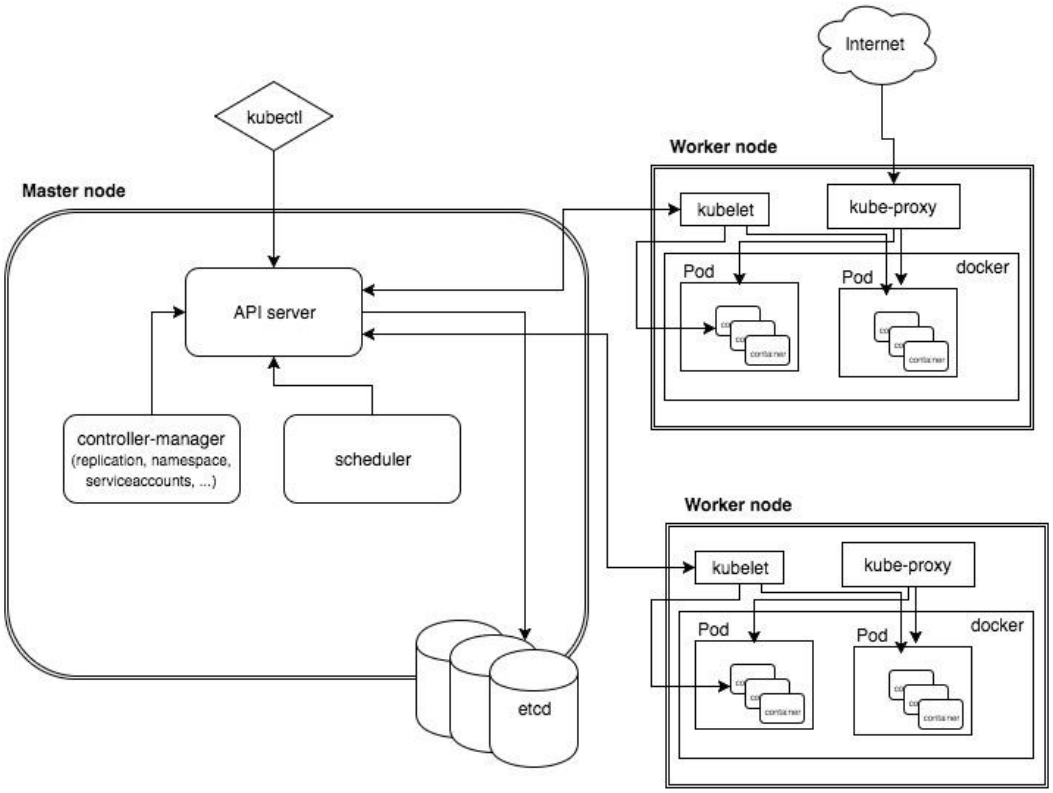
برای مدیریت کانتینرها روش‌ها و ابزارهای گوناگونی وجود دارند، یکی از این ابزارها که شهرت بسیار زیادی دارد Kubernetes [9] می‌باشد. این ابزار یک ارکستراتور[۱۴]می‌باشد. وظیفه‌ی این ابزار مانیتور کردن و در صورت لزوم دوباره اجرا کردن کانتینرهایی است که از دسترس خارج شده‌اند. این ابزار می‌تواند در صورتی که بار روی یکی از کانتینرها زیاد شده باشد، نسخه‌های دیگری از آن را جهت توزیع بار ایجاد نماید، به این ترتیت می‌توان بار حاصل از اتصال تعداد بسیار زیادی از اشیا را میان کامپوننت‌هایی از یک جنس توزیع کرد.

---

[1] Container [2]

[1] Virtual Machines [3]

[1] Orchestrator [4]




**شکل ۴-۷ معماری Kubernetes [11]**

یکی از مولفه‌های تاثیرگذار در پروژه سرویس connectivity می‌باشد. همانطور که در فصل دوم نیز بیان شد به دلیل نیازمندی‌های خاص پروژه این سرویس توسط خودمان پیاده‌سازی شد. این سرویس لیست تمامی کامپوننت‌های موجود را نگه داشته و می‌تواند پیام‌ها را به صورت تصادفی به یکی از کامپوننت‌هایی که از یک نوع می‌باشند، بدهد. به این ترتیب این پلتفرم می‌تواند بدون نیاز به هیچ تنظیمات خاصی گسترش پیدا کرده و از تعداد بیشتری شی پشتیبانی کند.

# ۵

# فصل پنجم

# ارزیابی

# ارزیابی

برای ارزیابی آنچه در طی این پروژه پیاده‌سازی شد، تست‌هایی بر روی هر مولفه برای درستی عملکرد آن صورت گرفت ولی مهمترین تست این پروژه تستی بود که به صورت کامل تمامیت پلتفرم و اجزای آن را مورد آزمایش قرار داد.

برای تست درستی عملگرد پلفترم به صورت کامل از یک حسگر چندکاره[۵]استفاده در یک محیط تست استفاده کردیم و داده‌های حاصل از آن را جمع‌آوری کردیم. حسگر چندکاره‌ی موجود دما، رطوبت و شدت نور را اندازه‌گیری می‌کند و آن را در بازه‌های ۵ دقیقه‌ای با استفاده از شبکه nRF برای master ارسال می‌کند، master داده‌ی دریافتی از طریق پورت سریال در اختیار agent قرار داده و agent نیز آن را برای پلفترم از طریق پشته‌ی پروتکلی پلتفرم ارسال می‌کند. در ادامه این روند درستی‌سنجی را به وسیله‌ی اشکال مرور می‌کنیم.

**شکل ۵-۱ سنسور چندگانه‌ی مبتنی بر شبکه nRF**

[1] Multi-sensor [5]

شکل ۵-۲ master شبکه nRF

```
::
 parham @ parham-USVM-1 ❯ ◇/Kaktos ❯ master
± node index.js
7 3290 [ { name: 'temperature', id: '1', value: '26' },
  { name: 'humidity', id: '1', value: '26' },
  { name: 'light', id: '1', value: '353' },
  { name: 'motion', id: '1', value: '0' } ]
7 3290 [ { name: 'temperature', id: '1', value: '26' },
  { name: 'humidity', id: '1', value: '26' },
  { name: 'light', id: '1', value: '354' },
  { name: 'motion', id: '1', value: '0' } ]
```

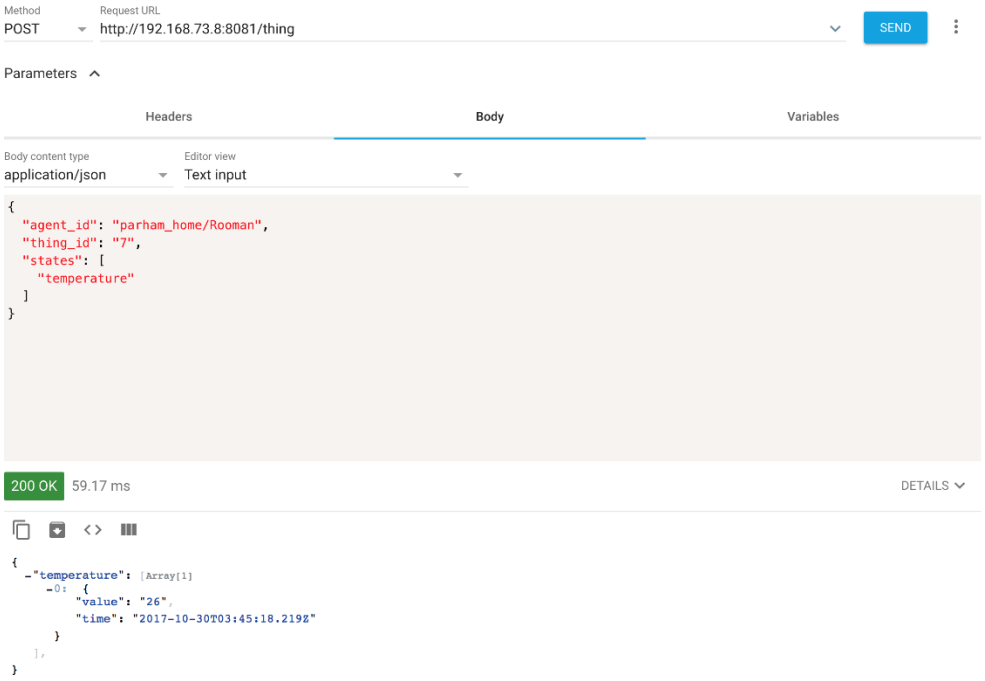
شکل ۵-۳ agent کاکتوس در حال دریافت داده‌های حسگرچندکاره از master

```
::
 parham @ bambil-vm ❯ ◇/bamboo/connectivity ❯ master
± node index.js
 * 18.20 at Sep 07 2016 7:20 IR721
 * MQTT 0.0.0.0:1883 on 1
 * MQTT 0.0.0.0:1883 on 2
Bamboo - Connectivity >
```

شکل ۵-۴ مولفه‌ی connectivity در حال اجرا


**شکل ۵-۵ مولفه‌ی log در حال جمع‌آوری داده‌ها**


**شکل ۵-۶ پایگاه‌داده‌ای influx در حال اجرا**




**شکل ۵-۷ تقاضا برای دریافت آخرین داده‌ی دمای ارسالی از agent**

در ادامه پلتفرم می‌بایست در مقابل بار تست شود، برای این تست یک آزمون کننده طراحی شد که می‌تواند تعداد زیادی agent را شبیه‌سازی کند و به این وسیله بار زیاد را برای پلتفرم ایجاد کند. در این آزمون کننده هر agent یک پروسه‌ی مجزا می‌باشد که به این ترتیب هر یک از آن‌ها بار مستقل خود را تولید می‌کند. [15] نتایج این آزمون به شرح زیر است:

| vCPU | Memory |
|---|---|
| 6 | 12Gb |

| Nodes | Status |
|---|---|
| 1000 | Fail |

| Nodes | Status |
|---|---|
| 500 | Success |
| 50 | Success |

همانطور که در نتایج فوق مشهود است پلتفرم حاضر بر روی یک بستر متوسط (۶ پردازنده و ۱۲ گیگابایت رم) می‌تواند حداکثر تا ۵۰۰ نود را بدون مشکل پشتیبانی کند. یکی از مشکلات موجود برای ارزیابی، توان پردازشی لازم برای آزمون کننده می‌باشد که برای شبیه‌سازی ۵۰۰۰ نود نیاز به سخت افزار قدرتمندی دارد.

۶

# فصل ششم

# جمع‌بندی و کارهای آینده

## جمع‌بندی و کارهای آینده

آنچه در طی این پروژه پیاده‌سازی شد، یک نسخه‌ی عملیاتی از یک پلتفرم عام منظوره می‌باشد که در طی چند ماه در محیط‌هایی مانند خانه، اتاق شورا دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر و ... تست گشت. بر روی این پلتفرم می‌توان برنامه‌هایی کاربردی جهت پردازش داده‌های جمع‌آوری شده، سناریوهای هوشمند و .. نوشت و به این ترتیب به گسترش آن کمک کرد. برای سهولت در این امر کتابخانه‌ای نوشته شده و در گیت‌هاب قرار داده شده است.

این پلتفرم به صورت سرویس محور نوشته شده است و می‌توان در آینده کاربردهایی را که نیاز به سرعت بیشتری در ارتباطات دارند را به صورت مولفه‌های جدید در آن پیاده‌سازی کرد.

مدیریت کاربران در این پلتفرم صورت نمی‌پذیرد و می‌توان در آینده این ویژگی را نیز به آن اضافه کرد که بتوان برای سیستم کاربر تعریف کرده و و برای کاربران سطح دسترسی تعیین کرد.

همانطور که در نیازسنجی‌های امنیتی نیز اشاره شد، یکی از جنبه‌های مهم مدیریت هویت اشیا متصل شده است و در این پلتفرم این امر توسط یک شناسه‌ی یکتا که به agentها تعلق می‌گیرد صورت می‌پذیرد. این مهم می‌تواند در نسخه‌های بعدی بهبود پیدا کند. یکی از این روش‌های بهبود استفاده از روش‌هایی است که در پلتفرم Kaa استفاده شده است، به این صورت که از یک کلید برای تشخیص هویت agent استفاده می‌شود که این کلید توسط کاربر بر روی agent قرار می‌گیرد.

# منابع و مراجع

[1] M. Richards, Software Architecture Patterns, O'Reilly Media, 2015.

[2] P. Alvani, "Bamboo-Connectivity," Bambil, 2017. [Online]. Available: https://github.com/bambil/bamboo-connectivity.

[3] "IoT Platforms: The central backbone for the Internet of Things," IoT Analytics, 2015.

[4] "Kaa," KaaIoT, 2017. [Online]. Available: https://www.kaaproject.org/. [Accessed 5 10 2017].

[5] S. Khatri, "Jasper Blog: IoT Security Checklist," Cisco, 14 March 2017. [Online]. Available: http://blog.jasper.com/iot-security-checklist/.

[6] R. J. Cohn, MQTT Version 3.1.1, OASIS Standard, 2015.

[7] P. Alvani, "Bamboo-Component," Bambil, 2017. [Online]. Available: https://github.com/bambil/bamboo-component.

[8] P. Alvani, "Bamboo-Log," Bambil, 2017. [Online]. Available: https://github.com/bamboo-log.

[9] G. Rauch, "Socket IO," 2017. [Online]. Available: https://socket.io.

[10] P. Alvani, "Bamboo-Trap," Bambil, 2017. [Online]. Available: https://github.com/bambil/bamboo-trap.

[11] P. Alvani, "socket.io-perfomace," Bambil, 2017. [Online]. Available: https://github.com/bambil/socket.io-perfomace.

[12] P. Alvani, "Bamboo-Set," Bambil, 2017. [Online]. Available: https://github.com/bambil/bamboo-set.

[13] P. Alvani, "Bamboolia," Bambil, 2017. [Online]. Available: https://github.com/bambil/bamboolia.

[14] T. K. Authors, "Production-Grade Container Orchestration," Cloud Native Computing Foundation, [Online]. Available: https://kubernetes.io/.

[15] P. Alvani, "Bamboo-Tester," Bambil, 2017. [Online]. Available:

https://github.com/bambil/bamboo-tester.

[16] N. Isabekyan, "Introduction to Kubernetes Architecture," X-Team, 11 Jul 2016. [Online]. Available: https://x-team.com/blog/introduction-kubernetes-architecture/.

**Amirkabir University of Technology**
**(Tehran Polytechnic)**

**Computer and Information Technology Engineering Department**

**B.Sc. Thesis**

**Title**

# Implementing an IoT Platform

**By**
**Parham Alvani**

**Supervisor**
**Dr. Masoud Sabaei**

**October 2017**