|  |  |  |
| --- | --- | --- |
| Description: Description: بازنشسته.jpg | ارتش جمهوری اسلامی ایران (فرماندهی جنگال و سایبر راهبردی آجا ) |  |

عنوان طرح:

طراحی و پیادهسازی سامانه هوشمند نظارت بر تصاویر دوربین های پیرامونی تحت شبکه

نام و نام خانوادگی مجری طرح:

حسین غلامی

استاد راهنما

نام کامل استاد راهنما

استادمشاور

نام کامل استاد مشاور

پاییز 1400

به نام خدا

صفحه تعهد نامه

چكيده

امنیت یکی از مفاهیم اولیه ای است که انسان از ابتدا با آن درگیر بود و همواره در صدد افزایش آن است. یکی از مواردی که در سال های اخیر برای افزایش امینیت رونق گرفته و پیشرفت کرده، ایجاد امنیت توسط سیستم­های نظارتی و دوربین هاست. دوربین ها نمیتوانند به صرف ذخیره اطلاعات امنیت را تامین کنند، و برای ایجاد امنیت نیازمند آن هستیم که شخصی بر دوربین ها نظارت دائمی داشته باشد که فرآیندی هزینه بر است و احتمال وجود خطای انسانی وجود دارد. در این پروژه هدف پیاده سازی سیستمی است که بتواند به کمک شخص ناظر آمده و فرایند مانیتور کردن دوربین ها را با کمک هوش مصنوعی و پردازش در یک سرویس دهنده تسهیل کند. در این پروژه سه برنامه کاربردی ارائه شده، به این نحوه که یک برنامه ای بر روی کامپیوتر سرویس دهنده ارائه شده که اطلاعات دوربین ها را دریافت و فرایند مدریت کاربران را انجام دهد، برنامه ای بر روی کامپیوتر سرویس دهنده دیگری اجرا شده که پردازش­های مربوط به هوش مصنوعی را انجام دهد، و برنامه ای برای کاربران ارائه شده که بتوانند فرایند دسترسی به اطلاعات دوربین ها را برعهده بگیرند. همچنین در این پروژه برای دوربین ها سطح دسترسی تعریف شده که کاربران بر اساس سطح دسترسی خود به دوربین مذکور دسترسی پیدا کنند. و البته اطلاعات دوربین ها در کامپیوتر سرویس دهنده ذخیره شده که کاربران بتوانند به اطلاعات ذخیره شده دسترسی پیدا کنند. برای این امر از سه سرویس RabbitMQ ، MinIO ، Redis ، PyQt استفاده شد. از RabbitMQ برای به جریان انداختن اطلاعات به صورت زمان واقعی ، از MinIO برای ذخیره سازی اطلاعات دوربین ها و دسترسی به آن ها در شبکه ، از Redis برای همگام سازی سه برنامه کاربردی با یکدیگر و از PyQt برای پیاده سازی ظاهر برنامه های کاربردی استفاده شد.

واژه‌های کلیدی:

VideoStraming,RabbitMQ,MinIO,Docker,PyQt

|  |  |
| --- | --- |
| فهرست مطالب | صفحه |

[1 فصل اول مقدمه 5](#_Toc90319291)

[1-1 پیشگفتار و اهمیت موضوع 6](#_Toc90319292)

[2-1 کار های پیشین انجام شده در این زمینه 7](#_Toc90319293)

[3-1 طرح مسئله 7](#_Toc90319294)

[4-1 ساختار پایان نامه 8](#_Toc90319295)

[2 فصل دوم سرویس های مورد استفاده 9](#_Toc90319298)

[1-2 مفهوم یکپارچهسازی 10](#_Toc90319299)

[2-2 مروری بر داکر 13](#_Toc90319300)

[3-2 مروری بر کارگزار RabbitMQ 17](#_Toc90319301)

[1-3-2 راهاندازی RabbitMQ با داکر 22](#_Toc90319302)

[4-2 مروری بر ذخیره کننده Redis 23](#_Toc90319303)

[1-4-2 راهاندازی Redis با داکر کامپوز 26](#_Toc90319304)

[5-2 پیاده‌سازی سیستم دریافت و ذخیره‌سازی 26](#_Toc90319305)

[6-2 جمع بندی 29](#_Toc90319306)

[3 فصل سوم طراحی و معماری برنامه های کاربردی 30](#_Toc90319307)

[1-3 گوجه فروشی حسین 31](#_Toc90319308)

[2-3 خیار فروشی محمد 31](#_Toc90319309)

[3-3 جمع بندی 31](#_Toc90319310)

[4 فصل چهارم بکارگیری و نحوه کارکرد سیستم برای کاربران 32](#_Toc90319311)

[1-4 راه اندازی و پیشنیاز ها 33](#_Toc90319312)

[2-4 جمع‌بندی فصل چهارم 33](#_Toc90319313)

[5 فصل هفتم نتیجه‌گیری و پیشنهادها 34](#_Toc90319314)

[1-5 نتیجهگیری 35](#_Toc90319316)

[2-5 پیشنهادها 36](#_Toc90319317)

[6 مراجع 37](#_Toc90319318)

|  |  |
| --- | --- |
| فهرست اشكال | صفحه |

[شکل ‏4–1-مثالی از یکپارچه‌سازی مبتنی بر فایل 11](#_Toc90319227)

[شکل ‏4–2-مثالی از یکپارچهسازی مبتنی بر پایگاهداده 12](#_Toc90319228)

[شکل ‏4–3-نمونه از یکپارچهسازی بر اساس ارتباط مستقیم میان برنامه‌ها 12](#_Toc90319229)

[شکل ‏4–4-نمونه‌ای از یکپارچهسازی با کمک یک کارگزار پیام 13](#_Toc90319230)

[شکل ‏4–5-موتور و هسته داکر که روند فرایند مدیریت برنامه را مشخص میکند. [22] 15](#_Toc90319231)

[شکل ‏4–6- نمایی از معماری داکر و فرایند چرخش دستورات و اطلاعات. [22] 17](#_Toc90319232)

[شکل ‏4–7-نمونه‌ای از یک فایل dockerfile 17](#_Toc90319233)

[شکل ‏4–8- نمونه‌ای از یک فایل docker-compose.yml 18](#_Toc90319234)

[شکل ‏4–9-معماری کلی نرم‌افزار RabbitMQ [23] 19](#_Toc90319235)

[شکل ‏4–10- داشبورد مدیریتی نرم‌افزار RabbitMQ 20](#_Toc90319236)

[شکل ‏4–11-انتقال پیام در نرم‌افزار RabbitMQ [23] 21](#_Toc90319237)

[شکل ‏4–12-نحوه استفاده از کلیدمسیریابی برای صف‌ها 22](#_Toc90319238)

[**شکل ‏4–13-اتصال دو مصرف‌کننده‌ی اطلاعات به یک صف** 22](#_Toc90319239)

[شکل ‏4–14-محتوای داکر فایل برای راه‌اندازی RabbitMQ 23](#_Toc90319240)

[شکل ‏4–15- بخش مرتبط با RabbitMQ در داکر کامپوز 23](#_Toc90319241)

[شکل ‏4–16- بررسی دستورات کار کردن با هشها در Redis 26](#_Toc90319242)

[شکل ‏4–17- بخش مرتبط با Redis در داکر کامپوز 27](#_Toc90319243)

[شکل ‏4–18- فرایند انتقال بسته‌ها از بخش توزیع دادگان به صفها 28](#_Toc90319244)

[شکل ‏4–19-کلاس مربوطه به برنامه انتقال‌دهنده اطلاعات RabbitMQ به Redis 29](#_Toc90319245)

[شکل ‏4–20-تابع فراخوانی اطلاعات هنگام دریافت اطلاعات از Redis و انتقال به RabbitMQ 30](#_Toc90319246)

|  |  |
| --- | --- |
| فهرست جداول | صفحه |

**No table of figures entries found.**

# فصل اول مقدمه

## پیشگفتار و اهمیت موضوع

تعریف فرهنگ لغات از امنیت, عبارت است از: در معرض خطر نبودن یا از خطر محافظت شدن. امنیت همچنین عبارت است از: رهایى از تردید, آزادى از اضطراب و بیمناکى و داشتن اعتماد و اطمینان موجه و مستند. امنیت, خواه فردى, ملى یا بین المللى, در زمره مسایلى است که انسان با آن مواجه مى باشد. امنیت به صورت وسیع, در مفهومى به کار گرفته شده که به صلح, آزادى, اعتماد, سلامتى و دیگر شرایطى اشاره مى کند که فرد و یا گروهى از مردم, احساس آزادى از نگرانى, ترس, خطر یا تهدیدات ناشى از داخل یا خارج را داشته باشند. هر چند آن گونه که بارى بوزانBozan) ) استدلال مى کند, مفهوم امنیت فى نفسه در کاربرد عامى که در روابط بین الملل و دیگر رشته ها دارد و به نظر مى رسد که به عنوان یک مفهوم سازمان یافته محورى از سوى سیاستمداران و محققان مورد پذیرش قرار گرفته باشد, با ادبیات شدیدا نامنسجمى روبروست. بوزان استدلال مى کند که بخش وسیع و موفق مطالعات موجود در بخش تجرى به مسایل و موضوعات امنیت ملى معاصر مربوط مى گردد. بیش تر این آثار از گرایش مطالعات استراتژیک استخراج شده است. به همین دلیل, امنیت بر محوریت هنجارى تمرکز یافت. فصل مشترک سیاست هاى خارجى, اقتصادى و نظامى کشورها, در حوزه هاى متغیر یا مورد منازعه و ساختار عام روابطى که آنها به وجود مىآورند, براى تحصیل امنیت ملى یا بین المللى, همه در اصطلاح ((آرمان ها)) تجزیه و تحلیل شده اند. به هر حال تا ظهور نگرانى هاى اقتصادى و محیطى در خلال دهه 1970, مفهوم امنیت به ندرت به غیر از پالیسى (سیاست) منافع بازیگران خاصى اشاره مى کرد و تا اواخر دهه 1980 در مباحث امنیت محوریت بعد نظامى هنوز مورد تاکید بود. [1]

استفاده از دوربین های نظارتی جهت امنیت خیلی خوبه

استفاده از دوربین های نظارتی در همه محل ها خیلی خوبه ولی برای ارگان های نظامی مثل نون شب واجبه

حالا دوربین بدون نظارت و مانیتورینگ که معنی نداره پس باید همواره یکی پای دوربین باشه و دوربین رو نگاه کنه این میتونه منجر به خطای انسانی بشه و اونی که داره دوربین رو نگاه میکنه همیشه حواسش نباشه پس باید بتونیم به اون شخص که داره دوربین ها رو نگاه میکنه با کمک هوش مصنوعی کمک کنیم و سیستمی رو پیاده کنیم که بشه از الگوریتم های هوش مصنوعی که دنیا داره رو اون سرمایه گذاری میکنه توی سیستم های خودمون استفاده کنیم

## کار های پیشین انجام شده در این زمینه

در دنیا شرکت هایی هستند که این سرویس رو ارائه میدن چنتا از اون ها رو با هم مرور میکنیم.

مثلا سایت avigilon و .... متاسفانه در کشور عزیزمون ایران شرکتی در این زمینه فعال نیست.

## طرح مسئله

بهتر است مسئله‌ای را در همین ابتدا با بررسی یکی از اصلی­ترین چالش­های پروژه شروع کنیم تا درک بهتری از مسئله پیش رو داشته باشیم. این چالش را با یک مثال ساده و با اعداد بررسی می­کنیم؛ در تهران بر اساس [1] بیش از 4 میلیون خودرو درحرکت‌اند. با فرض نمونه­گیری از حسگرها با فرکانس 50 هرتز برای تشخیص رویدادها و آماده کردن بسته‌های 1 کیلوبایتی، حجم ارسال اطلاعات در ثانیه،200 گیگابایت است که برای ذخیره‌سازی، پردازش و ... حجم زیادی است. پس واضح است که با یک مسئله دادگان انبوه مواجه هستیم. اگرچه می­توان این حجم از اطلاعات را کنترل نمود ولی همواره به دنبال راهکارهای ساده‌تری هستیم تا جایگزین شیوه فوق شود. در این پژوهش از شیوه دیگری استفاده شد، به این شکل که روند تشخیص رویدادهای خطرناک، به داخل هر خودرو منتقل شود و با راهکاری بتوان در زمان واقعی، رویدادها را تشخیص داد و به‌جای همه‌ی اطلاعات تنها رخدادهای خطرناک را به ابر منتقل کرد. پس با توجه به توضیح فوق، برای تحقق هدف اصلی پروژه که پیاده­سازی روشی برای تشخیص رویدادهای رانندگی و تجمیع و بررسی اطلاعات رانندگان در ابر محاسباتی است، پروژه به سه بخش متفاوت تقسیم شد که به شرح زیر است:

* طراحی مدلی که از اطلاعات سنسوری رخدادهای خطرناک رانندگی را تشخیص دهد.
* پیاده‌سازی محیطی در ابر که عمل دریافت و ذخیره‌سازی اطلاعات را به انجام رساند.
* طراحی یک مدل آماری که رانندگی افراد را مقایسه کند و به رفتارهای رانندگان امتیاز دهد.

اطلاعات سنسوری به‌کاررفته در این پژوهش اطلاعات حسگرهای شتاب­سنج و ژیروسکوپ است که هر دو حسگرهای ارزان‌قیمتی محسوب شده و احتمال خراب شدن آن‌ها بسیار کم است که با بهره­گیری از اطلاعات آن­ها توانستیم رخدادهای خطرناک رانندگی ازجمله؛ گردش به چپ و راست شدید، ترمز و شتابگیری شدید، تعویض لاین به چپ و راست شدید را با دقت قابل قبولی تشخیص دهیم. این فرایند با آموزش یک مدل درخت تصمیم­گیری[[1]](#footnote-1) که بردارهای ویژگی آن از یک پنجره­بندی اتفاقی و الگوریتم Fast-DTW بدست آمده، بر روی یک مجموعه دادگان که برچسب‌های رخدادهای فوق را داشتند، انجام شد. در ادامه با کمک Docker، با به‌کارگیری برنامه‌های RabbitMQ، Redis و برنامه­ای میانی که این دو نرم‌افزار را به هم متصل کند سیستمی یکپارچه بر بستر ابری طراحی شد تا به‌عنوان ابر محاسباتی وظیفه دریافت و ذخیره­سازی اطلاعات را انجام دهد. در پایان نیز با کمک مجموعه دادگانی دیگر که حجم اطلاعات بیشتری در اختیار ما قرار می­داد یک مدل مبتنی بر توزیع‌های آماری طراحی شد تا با کمک آن بتوانیم نحوه عملکرد رانندگان را تشخیص دهیم و آن‌ها را مورد ارزیابی قرار دهیم.

## ساختار پایان نامه

در این رساله، که در هفت فصل تدوین شده، در ابتدای هر فصل ابتدا مروری بر مفاهیم کلی، الگوریتم‌ها و نرم‌افزارهای به‌کاررفته، انجام شده و در ادامه جزئیات در بخش­ها و طراحی آن را شرح داده خواهد شد. در فصل دوم به مروری بر کارهای انجام شده در این زمینه اشاره میشود و س در فصل سوم به طراحی مدلی که رخدادهای خطرناک را تشخیص می­دهد، پرداخته شده، در فصل چهارم به پیاده­سازی ابرمحاسباتی که وظیفه دریافت و ذخیره­سازی را دارد، اشاره شده، در فصل پنجم نحوه امتیاز دهی به رانندگان توضیح داده شده، در فصل ششم، عملکرد هر یک از بخش­ها به طور کامل بررسی می­شود. در فصل هفتم نیز به نتیجه­گیری و پیشنهادهایی که در ادامه راه این پروژه می­تواند انجام شود، پرداخته می­­شود.



# فصل دوم سرویس های مورد استفاده

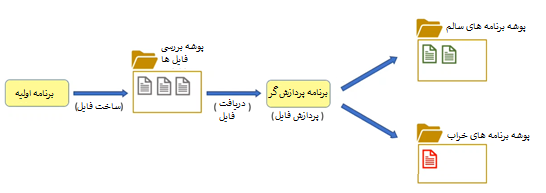
**پیاده­سازی سیستم دریافت و ذخیره­سازی اطلاعات**

در این بخش به پیاده‌سازی بستری برای دریافت اطلاعات می­پردازیم و سعی می­کنیم با ابزارهای یکپارچه‌سازی بتوانیم، شرایطی برای دریافت و ذخیره‌سازی اطلاعات فراهم کنیم. در ابتدا مفهوم یکپارچه‌سازی و سپس تعدادی از نرم‌افزارهای مورداستفاده را بررسی کرده و در ادامه نحوه استفاده و ارتباط آن‌ها با یکدیگر شرح داده می­شود.

## مفهوم یکپارچه­سازی

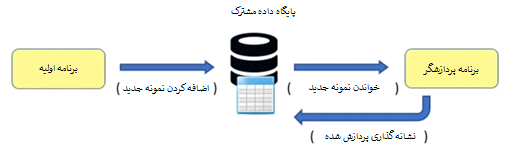
یک سیستم کامل، از چند برنامه تشکیل‌شده که وظایف متفاوتی را انجام می­دهند، در اغلب مواقع این برنامه‌ها نیاز دارند با یکدیگر تعامل داشته باشند و باهم ارتباط برقرار کنند. به ایجاد کردن ارتباط میان برنامه‌های متفاوت یکپارچه‌سازی[[2]](#footnote-2) می­گویند. شیوه‌های متفاوتی از یکپارچه­سازی وجود دارد.

مدل اول یکپارچه­سازی مبتنی بر فایل است. در این مدل، برنامه اولیه فایلی که باید پردازش شود را ایجاد می­کند و در پوشه‌ای مشخص قرار می­دهد، سپس برنامه‌ای دیگر فایل‌هایی که در این پوشه قرارگرفته‌اند را بررسی می­کند. برای مثال برنامه دوم و فایل­های خراب را از سالم جدا کرده و در پوشه‌هایی متفاوت قرار می­دهد. قابل‌مشاهده است که این دو برنامه می­توانند مستقل از هم کار کنند و حتی می­توانند با زبان‌های برنامه‌نویسی متفاوتی ایجادشده باشند. در شکل ‏4–1 می­توان مثال فوق را مشاهده کرد.



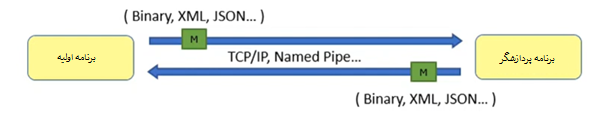
شکل ‏4–1-مثالی از یکپارچه‌سازی مبتنی بر فایل

مدل دوم یکپارچه­سازی­هایی مبتنی بر پایگاه­داده مشترک یا توزیع‌شده است. در این مدل ابتدا یکی از برنامه­ها اطلاعاتی را در پایگاه­داده تغییر می­دهد. سپس برنامه دیگری آن را برداشته و پردازش می­کند و مجدداً به پایگاه­داده برمی­گرداند و به آن برچسب پردازش‌شده می­زند. به‌طور مشابه، این دو برنامه می­توانند مستقل از یکدیگر فعالیت کنند. در شکل 4–2 می­توان مدل فوق را بررسی نمود.



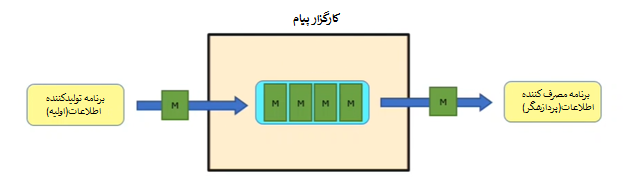
شکل ‏4–2-مثالی از یکپارچه­سازی مبتنی بر پایگاه­داده

مدل سوم یکپارچه­سازی­هایی مبتنی بر ارتباط مستقیم است، به‌نحوی‌که برنامه‌ها به‌صورت مستقیم با یکدیگر ارتباط tcp/ip برقرار کرده و بعد از برقراری ارتباط شروع به ارسال پیام به یکدیگر می­کنند. فرمت پیام‌ها می­تواند به گونه‌های متفاوتی ازجمله باینری یا مبتنی برنوشته مانند, xml, json ... باشد. در شکل ‏4–3 می­توان نمونه‌ای از این ارتباط را مشاهده کرد.



شکل ‏4–3-نمونه از یکپارچه­سازی بر اساس ارتباط مستقیم میان برنامه‌ها

مدل چهارم یکپارچه­سازی بر پایه پیام دادن نامتقارن با کمک یک کارگزار پیام[[3]](#footnote-3) است. برنامه اولیه یا تولیدکننده اطلاعات در یک‌سو با هر فرمتی که بخواهد می­تواند، پیام ارسال کند ولی این باریک برنامه میانی پیام‌ها را دریافت می­کند (که اغلب به آن message broker یا message bus می­گویند) پیام‌ها را در یک لیست قرار داده که به آن Queue یا صف می­گویند و آن‌ها را به هدف، مقصد یا مصرف‌کننده منتقل می­کند. در شکل ‏4–4 می­توان نمونه‌ای از این شیوه انتقال اطلاعات را مشاهده کرد. این روش، یک روش محبوب است که در سیستم­های متفاوتی می­تواند مورداستفاده قرار بگیرد.



شکل ‏4–4-نمونه‌ای از یکپارچه­سازی با کمک یک کارگزار پیام

این شیوه از یکپارچه‌سازی مزایای بسیاری دارد که به شرح زیر است:

* جدا بودن برنامه تولیدکننده از برنامه مصرف‌کننده:
  + برنامه تولیدکننده اطلاعات و برنامه مصرف‌کننده نیازی ندارد یکدیگر را بشناسند.
  + آدرس و تکنولوژی‌های مورداستفاده یکدیگر را نمی­دانند.
  + تنها چیزی که نیاز است نسبت به هم آگاه باشند، فرمت اطلاعات ارسالی و دریافتی است.
* ایجاد محیطی مطمئن، قابل‌اعتماد برای انجام پردازش‌ها و ارتباط میان برنامه­ها:
  + برنامه‌های تولیدکننده اطلاعات می­تواند به ارسال اطلاعات بپردازد، درحالی‌که هیچ برنامه‌ای در طرف دیگر، اطلاعات را مورداستفاده قرار نمی­دهد، به این معنا که اطلاعات در برنامه میانی به‌صورت موقت ذخیره می­شوند و در ادامه هرگاه که برنامه مصرف‌کننده اطلاعات، فعالیت خود را آغاز کرد، می­تواند از اطلاعات استفاده نماید.
  + درصورتی‌که برنامه مصرف‌کننده در پردازش شکست بخورد، می­تواند درخواست ارسال مجدد داشته باشد.
* فراهم کردن راهی برای پیاده­سازی معماری­هایی افقی در تعداد بالا
  + اگر برنامه مصرف‌کننده اطلاعات به‌تنهایی نتواند همه پیام‌ها را پردازش کند، می­توان چندین مصرف‌کننده اطلاعات را در کنار هم قرارداد تا فرایند سریع­تر انجام شود.

این شیوه بسیار پربازده‌تر از مدل‌های یکپارچه­سازی مبتنی بر پایگاه­داده، است.

## مروری بر داکر

به‌طورکلی داکر[[4]](#footnote-4) یک سکو برای توسعه، انتقال و اجرای برنامه‌ها است. با کمک داکر می­توانیم نرم­افزار­ها را مستقل از زیرساخت به‌سرعت پیاده‌سازی کنیم. همچنین با کمک این نرم‌افزار می­توان زیرساخت و سخت‌افزار مورداستفاده در پروژه‌ها را مدیریت کرد و فاصله زمانی میان پیاده‌سازی و رسیدن به مرحله محصول نهایی را به حداقل رساند.

داکر توانایی بسته‌بندی و اجرای برنامه­های ما را در محیط ایزوله‌ای به نام کانتینر[[5]](#footnote-5) فراهم می­کند. این ایزوله شدن شرایطی ایجاد می­کند که کانتینرها را بتوان به‌صورت هم‌زمان روی کامپیوتر میزبان اجرا کرد. این کانتینرها در مقابل ماشین‌های مجازی بسیار سبک هستند و می­توانند بدون نظارت بر روی کامپیوتر میزبان اجرا شوند. به‌طورکلی داکر ابزار و سکویی برای مدیریت کردن این کانتینرها است تا توسعه سریع و ساده­تر شود و بتوان پردازش قابل توسعه­ای، روی خوشه‌ای از کامپیوترها اجرا کرد. همچنین هنگامی‌که برنامه کامل شد و به مرحله محصول نهایی رسید، بدون در نظر گرفتن نوع زیرساخت، قابل‌انتقال به محیط‌های دیگر (مرکز داده‌ها، شرکت‌های خدمات دهنده ابری،...) باشد.

به‌طورکلی هسته اصلی داکر از سه بخش تشکیل‌شده که در شکل ‏4–5 قابل‌مشاهده است، بخش اول یک سرویس‌دهنده است که مدیریت اشیا ساخته‌شده توسط داکر را بر عهده دارد، اشیائی همچون فایل ایمیج (فشرده‌شده کانتینرها)، کانتینرها، شبکه‌ها و فضاهایی که تخصیص داده می­شوند. این برنامه فرایند محاسباتی[[6]](#footnote-6) نام‌گذاری می­شود. بخش دوم یک برنامه رابط است (REST API) که دستوراتی که ما می­دهیم را به فرایند محاسباتی منتقل می­کند. بخش سوم یک رابط و خط دستور (CLI)[[7]](#footnote-7) است که فرمان‌های ما در آن وارد می­شود. خط دستور از یک REST API استفاده می­کند تا وظیفه کنترل و ارتباط با فرایند محاسباتی را یا با استفاده از فرمان­هایی از پیش نوشته‌شده یا به‌صورت واردکردن در خط دستور انجام دهد.



شکل ‏4–5-موتور و هسته داکر که روند فرایند مدیریت برنامه را مشخص می­کند. [2]

به سه دلیل اصلی در این پروژه برای پیاده‌سازی فرایند دریافت و ذخیره‌سازی اطلاعات از داکر استفاده شد که به شرح زیر است:

پیشرفت و توسعه مداوم برنامه‌ها؛

داکر با کار در محیط‌های استاندارد و با استفاده از کانتینرهای محلی این اجازه را به توسعه‌دهندگان برای پیاده­سازی برنامه‌های پیچیده می­دهد و چرخه عمر توسعه را ساده می­کند. چراکه بخش‌های مختلف برنامه می­توانند مستقل از هم با نسخه‌ها و حتی زبان­های برنامه‌نویسی متفاوت پیاده­سازی و اجرا شوند. به این عمل فرایند یکپارچه‌سازی پیوسته[[8]](#footnote-8)/ تحویل پیوسته[[9]](#footnote-9) می­گویند.

بستر مبتنی بر کانتینرهای داکر، ایجاد برنامه‌هایی قابل‌حمل[[10]](#footnote-10) را امکان‌پذیر می‌کند. کانتینرهای داکر می‌توانند بر روی کامپیوتر محلی توسعه‌دهنده، در ماشین‌های فیزیکی یا مجازی در یک مرکز داده، بر روی ارائه‌دهندگان خدمات ابری یا در مخلوطی از این محیط‌ها اجرا شوند که کمک می­کند برنامه به‌سادگی روی کامپیوتر توسعه‌دهنده آماده و سپس برای پیاده‌سازی عملی به سرورهایی باقدرت و پاسخ‌گویی بیشتر منتقل شود.

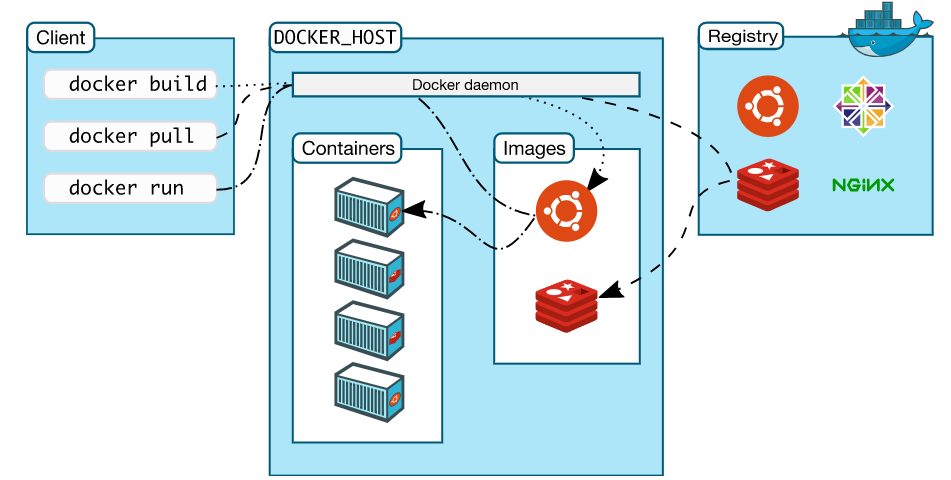
کانتینرهای داکر سبک و سریع هستند چراکه تنها برنامه‌هایی که موردنیاز است را در خود جای‌داده‌اند. این‌یک جایگزین مناسب و مقرون‌به‌صرفه به‌جای ماشین‌های مجازی است که نیازمند پیش‌نیازهای بیشتری برای پیاده‌سازی محیط‌های ایزوله هستند؛ بنابراین می­توان از ظرفیت محاسبه بیشتری برای رسیدن به اهداف خود استفاده کرد. داکر برای برنامه‌هایی با چگالی بالا (بخش‌های متفاوت) و عملکردهای کوچک و متوسط که نیاز به انجام کارهای بیشتر با منابع کمتری دارند، بسیار مناسب است.

داکر از یک معماری کاربر-سرویس­دهنده­ای استفاده می­کند. به این شکل که کاربر داکر، با فرایند محاسباتی در ارتباط است که وظیفه ساخت و اجرا و توزیع کانتینرها را بر عهده دارد. کاربر داکر و فرایند اصلی می­توانند در یک کامپیوتر باشند یا می­توانند از طریق REST API یا رابط­های شبکه با یکدیگر در ارتباط باشند. به‌طورکلی برنامه‌ها در محیطی به نام کانتینر اجرا می­شوند که نمونه‌ای قابل‌اجرا از فایلی به نام ایمیج است. برای ساخت فایل ایمیج یا باید خودمان آن فایل را از ابتدا ایجاد کنیم، یا از ایمیج های استانداری که در بخش رجیستری داکر (داکرهاب) قرارگرفته استفاده کنیم. عموماً برای سبک بودن محیط اجرای کار حتی درصورتی‌که بخواهیم فایل ایمیجی را خودمان طراحی کنیم از یک ایمیج مبتنی بر لینوکس ساده شروع کرده و هر چیزی که به آن نیاز داریم را به آن اضافه می­کنیم، ولی در اکثر مواقع برنامه‌های موردنیاز در داکرهاب وجود دارند که می­توان از آن بهره­برد.

برای مثال در شکل ‏4–6 با دستور اول یک فایل ایمیج ساخته می­شود. با دستور دوم یک فایل ایمیج از بخش رجیستری (داکرهاب) به کامپیوتر میزبان منتقل می­شود و با دستور سوم، یکی از فایل­های ایمیج ساخته‌شده به‌صورت کانتینر درآمده و اجرا می­شود.

به این شکل کانتینرها ایجاد می­شوند. حال برای ارتباط کانتینرها با یکدیگر از مکانیزم های دیگری که پیش­تر در شکل ‏4–5 دیدیم استفاده می­شود. مکانیزم هایی همچون ایجاد شبکه یا اجازه دسترسی به حجم‌های فیزیکی که ارتباط میان کانتینرها را فراهم کند.

همان‌طور که در شکل ‏4–6 قابل‌مشاهده است، برای اجرای هر مرحله می­بایست دستوری را اجرا کنیم تا مراحل پیش­رود و برنامه اجرا شود، این کار را می­توان با کمک داکر کامپوز به‌صورت خودکار انجام داد و روابط بین کانتینرها را مشخص کرد. داکر کامپوز یک ابزار برای تعریف و اجرای برنامه‌های حاوی چند کانتینر است. این برنامه از یک فایل YAML برای پیکربندی این مجموعه استفاده می­کند. سپس با اجرای تنها یک دستور همه‌ی سرویس­های خود را فعال و آماده‌به‌کار می­کند. ساخت برنامه‌ها با داکر کامپوز از سه بخش تشکیل‌شده است.



شکل ‏4–6- نمایی از معماری داکر و فرایند چرخش دستورات و اطلاعات. [2]

بخش اول ساخت Dockerfile است؛ که در حقیقت ایمیجی جدید بر اساس آنچه که در آن واردشده، می­سازد. برای مثال از ایمیج پایتون که به‌صورت استاندارد در رجیستری (داکرهاب) وجود دارد، شروع کرده و در آن فایل دستور مربوط به نصب کتابخانه‌های موردنیاز را وارد کرده، سپس برنامه پایتون نوشته‌شده را به کانتینر منتقل تا برنامه را در آن اجرا شود. در ادامه در شکل ‏4–7 نمونه‌ای از آن را مشاهده می­کنیم.

بخش دوم، سرویس‌هایی که است که در فایل docker-compose.yml تعریف می­شوند که شامل نحوه ارتباط کانتینرها با یکدیگر از طریق شبکه داخلی و ارتباط شبکه کانتینرها با شبکه خارجی (کامپیوتر میزبان) و تنظیم دسترسی­های حجمی بر روی کامپیوتر میزبان است. در شکل ‏4–8 نمونه‌ای از این فایل را مشاهده می­کنیم.

FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt

COPY main.py ./

COPY util.py ./

CMD [ "python", "./main.py" ]

شکل ‏4–7-نمونه‌ای از یک فایل dockerfile

networks:

  app-tier:

    driver: bridge

services:

  queue:

    build: rbmq/.

    container\_name: rabbitmq

    ports:

      - 5672:5672

      - 15672:15672

      - 1884:1884

    volumes:

      - "C:/Users/hosse/Docker/Rabbitmq/rbmq/Conf/:/etc/rabbitmq/"

    networks:

      - app-tier

  consumer:

    build: sl/.

    container\_name: stl

    volumes:

      - "C:/Users/hosse/Docker/Rabbitmq/sl/log/:/usr/src/app/log"

    networks:

      - app-tier

  cache:

    image: redis:latest

    container\_name: redis

    ports:

        - 6379:6379

    volumes:

        - C:/Users/hosse/Docker/Rabbitmq/redis/config/redis.conf:/redis.conf

        - C:/Users/hosse/Docker/Rabbitmq/redis/Data:/data

    command: [ "redis-server", "/redis.conf" ]

    networks:

      - app-tier

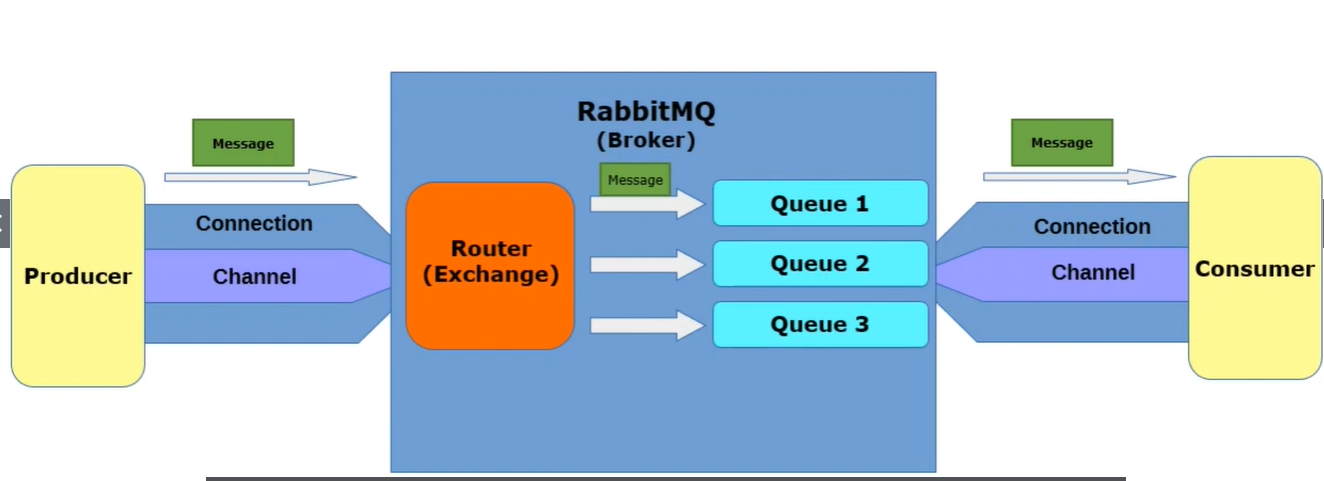
شکل ‏4–8- نمونه‌ای از یک فایل docker-compose.yml

در پایان هم تنها با اجرای دستور docker-compose -up، می­توان همه ایمیج­ها را از روی فایل docker-compose.yml، ساخته یا از رجیستری (داکرهاب) دریافت کرد و در ادامه از روی ایمیج­های ساخته‌شده، کانتینرها ساخته‌شده و برنامه­ها درون آن‌ها اجرا ­شوند.

## مروری بر کارگزار RabbitMQ

RabbitMq یک کارگزار پیام است که می­توان با کمک آن سیستم‌های بزرگی را یکپارچه­سازی کرد. به این شکل که پیام را ابتدا از تولیدکنندگان اطلاعات دریافت می­کند و به مصرف‌کنندگان اطلاعات منتقل می­کند. این نرم‌افزار متن‌باز و با زبان برنامه‌نویسی Erlang نوشته‌شده است و از پروتکل­های بسیاری ازجمله AMQP، STOMP، MQTT، HTTP و WebSocket پشتیبانی می­کند. دراین‌بین پروتکل MQTT، یکی از پرکاربردترین پروتکل‌ها در حوزه اینترنت اشیا است، چراکه بسیار سبک است و توانایی پیاده‌سازی بر روی سیستم‌های قابل توسعه را دارد. به همین جهت از این نرم‌افزار می­توان به‌عنوان پلی میان سخت‌افزار و محیط ابری یادکرد. این نرم‌افزار بر روی ویندوز، لینوکس و مک قابل‌استفاده است و همچنین به‌صورت کانتینر داکر در رجیستری داکر (داکرهاب) قرارگرفته است که استفاده از آن را بسیار ساده می­کند.

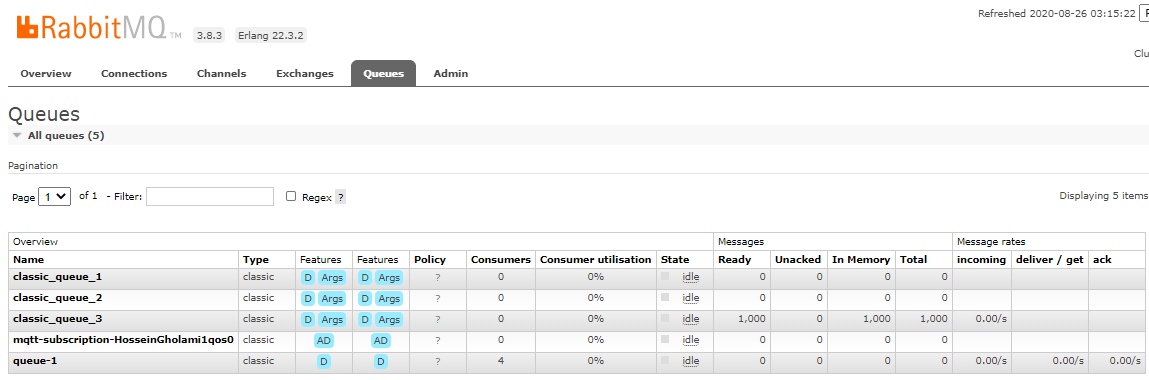
در شکل ‏4–9 معماری کلی این نرم‌افزار قابل‌مشاهده است، تولیدکنندگان اطلاعات[[11]](#footnote-11) و مصرف‌کنندگان اطلاعات[[12]](#footnote-12) در دو سمت این معماری و این نرم‌افزار در مرکز شکل قرارگرفته‌اند. آن‌ها به این نرم‌افزار متصل شده و به تبادل اطلاعات می­پردازند. در این معماری اجزای دیگری ازجمله صف[[13]](#footnote-13)، مرکز توزیع[[14]](#footnote-14) و کانال‌ها[[15]](#footnote-15) قابل‌مشاهده هستند که در ادامه با بررسی یک مثال به شرح وظایف هرکدام از این اجزا می­پردازیم.



شکل ‏4–9-معماری کلی نرم‌افزار RabbitMQ [3]

یک قطعه اینترنت اشیا را در نظر بگیرید که می­خواهد اطلاعات خود را با پروتکل MQTT به یک سرویس­دهنده ارسال کند. می­دانیم که این پروتکل از چندین کیفیت سرویس‌دهی پشتیبانی می­کند ولی با فرض استفاده از کیفیت سرویس‌دهی 1، یک بسته حاوی اطلاعات سنسور را به کارگزار پیام ارسال می­کند و صبر می­کند تا کارگزار دریافت این پیام را تصدیق کند. این ارتباط یک ارتباط TCP است و کانال که در شکل ‏4–9 تعریف شد، در حقیقت همان پورت مربوطه در هدر بسته‌های TCP است که این امکان را فراهم می­کند که از یک دستگاه، چندین ارتباط با کارگزار پیام به وجود آید. سپس این بسته دریافت شده و در مرکز توزیع بررسی می­شود. می­دانیم که بسته‌ها در پروتکل MQTT یک تاپیک دارند که گیرنده اطلاعات در مقصد با کمک آن مشخص می­­شود. پس در مرکز توزیع بسته­ها متناسب با تاپیک خود، در صف‌هایی که مربوط به خود هستند، قرار خواهند گرفت. در ادامه نحوه مدیریت مرکز توزیع به‌طور کامل شرح داده می­شود سپس بسته‌ها در صف‌هایی از پیش تعریف‌شده در نرم‌افزار ذخیره می­شوند. در این نرم‌افزار می­توان تنظیماتی دقیق برای هر صف انجام داد و جزئیاتی را در آن پیاده‌سازی کرد که در ادامه شرح داده می­شود. در پایان نیز یک مصرف‌کننده اطلاعات به این صف‌ها متصل می­شود که این بسته‌ها را از صف‌ها دریافت کرده و پردازش مربوطه را روی آن انجام می­دهد. یکی از ویژگی‌هایی که ما را به استفاده از RabbitMQ سوق می­دهد، این است که در سمت گیرنده اطلاعات دیگر نیازی به به‌کارگیری این پروتکل نیست و می­توان از پروتکل‌های دیگر که نرم­افزار پشتیبانی می­کند، استفاده نمود.

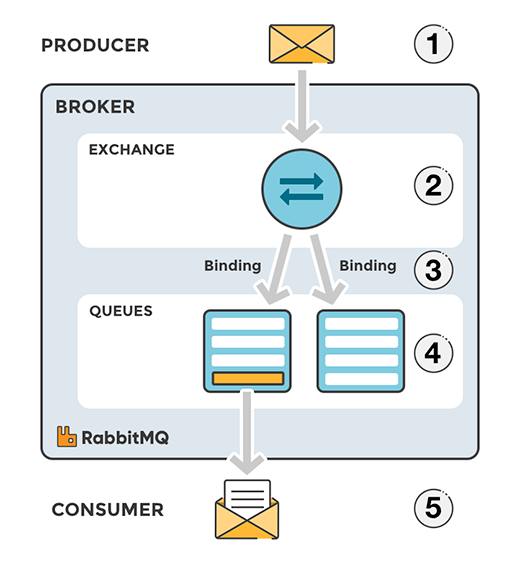
RabbitMQ به ما یک پنل داشبورد برای بررسی شرایط صف‌ها، تبادل داده‌ها و کانال‌ها و ارتباطات می­دهد که در آن می­توان اطلاعات مفیدی هنگام به‌کارگیری در عمل به دست آورد. در شکل ‏4–10 صفحه مربوط به‌صف­های را می­توانید مشاهده کنید، برای مثال در این تصویر 4 مصرف‌کننده اطلاعات بر روی صف queue-1 قرارگرفته شده تا در صورت انتقال اطلاعات به این صف، پیام‌ها دریافت شوند. درحالی‌که در صف classic\_queue\_3 هزار پیام در حافظه قرارگرفته و هیچ مصرف‌کننده اطلاعاتی به این صف متصل نیست.



شکل ‏4–10- داشبورد مدیریتی نرم‌افزار RabbitMQ

با توجه به توضیحات فوق و بررسی وظایف کلی هر بخش، به بررسی دقیق­تر انتقال پیام‌ها از مرکز توزیع وصف‌ها می­پردازیم.

در این پروژه برای سمت گیرنده اطلاعات از پروتکل MQTT استفاده شد به این معنا که بسته‌های حاوی اطلاعات سنسوری پس از دریافت توسط این پروتکل، به مرکز انتقال ex.mqtt منتقل می­شوند. می­دانیم بسته‌هایی که با این پروتکل ارسال می­شوند در خود بخشی با عنوان تاپیک دارند، RabbitMQ از این بخش هنگام تقسیم بسته‌ها در صف استفاده کرده و متناسب با مجموعه قوانینی قابل تنظیم بسته‌ها را در صف‌های مربوطه قرار می­دهد. در شکل ‏4–11 نمونه‌ای از این انتقال را می­بینیم.



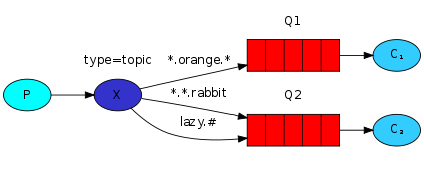
شکل ‏4–11-انتقال پیام در نرم‌افزار RabbitMQ [3]

هنگامی‌که هر صف تعریف می­شود، در خود بخشی با عنوان «کلید­ مسیریابی[[16]](#footnote-16)» دارد و باید به یک مرکز انتقال متصل شود. صف‌ها با کمک کلید انتقال بسته‌هایی که وارد، یک مرکز انتقال می­شوند را می­توانند در خود جای دهند. برای مثال stock.usd.ny و quick.orange.rabit می­توانند هر دو نام تاپیک برای بسته‌ها باشند و صف‌هایی با کلید مسیریابی فوق می­توانند، این پیام‌ها را دریافت کنند. دراین‌بین برای RabbitMQ از سه کاراکتر ‘.’ و ‘#’ و’\*’ برای مدیریت کردن بهتر بسته‌ها هنگام تقسیم بسته‌ها میان صف‌ها استفاده می­کند.

کاراکتر ‘.’ به معنی ایجاد ساختار است و هنگامی‌که در کلید مسیریابی استفاده شود باید دقیقا تاپیک پیام و کلیدمسیریابی یکسان باشد. کاراکتر ‘\*’ به این معنی است که در بخش‌هایی که این کاراکتر قرارگرفته می­تواند تاپیک بسته با کلیدمسیریابی متفاوت باشد و کاراکتر ‘#’ به این معناست، درصورتی‌که تا قبل از این کاراکتر یکسان بود، بسته به صف منتقل شود. مثال زیر را در نظر بگیرید:

بسته‌ای با تاپیک quick.orange.rabbit به یک مرکز انتقال می­رسد و چهار صف با کلیدهای مسیریابی زیر به این مرکز انتقال متصل هستند.

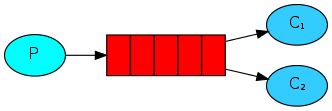
حالت اول کلیدمسیریابی quick.orange است، این پیام دریافت نمی­شود، چراکه تعداد بخش‌های ایجادشده با ‘.’ یکسان نیست. حالت دوم کلیدمسیریابی \*.orange.\* است، این پیام دریافت می­شود، چراکه تعداد بخش‌های ایجادشده با نقطه یکسان و از دو بخش اول و آخر بسته صرف‌نظر می­شود. حالت سوم، کلیدمسیریابی quick.# است، این پیام دریافت می­شود چراکه تاپیک بسته با هر آنچه که قبل از ‘#’ است، یکسان می­باشد. این مکانیزم می­تواند برای دسته‌بندی پیام‌های ورودی بسیار مؤثر باشد. در شکل ‏4–12 می­توان مثال دیگری را مشاهده نمود.



شکل ‏4–12-نحوه استفاده از کلیدمسیریابی برای صف‌ها

مکانیزم نگه­داری پیام‌ها در صف به‌طورکلی به دو صورت تقسیم‌بندی می­شود پیام‌های گذرا[[17]](#footnote-17) و پیام­های ثابت­شده[[18]](#footnote-18)، پیام‌ها، به‌طور عمومی در حافظه نوشته می­شوند (گذرا هستند) ولی درصورتی‌که حد آستانه‌ای که برای صف تنظیم‌شده بیشتر شوند پیام‌ها به ثابت‌شده، تبدیل شده و روی دیسک نوشته می­شوند و تنها یک اشاره­گر که محل ابتدای آدرس دیسک ثبت‌شده است، در حافظه نگهداری می­شود. هنگامی‌که مصرف‌کننده اطلاعات، درخواست اطلاعات می­کند، قبل از ارسال این پیام وارد حافظه شده و در حالت گذرا قرار می­گیرد و پس‌ازاینکه دریافت­کننده تصدیق انجام کار آن اطلاعات را ارسال کرد از حافظه نیز پاک می­شوند.

در این ساختار همان‌طور که پیش­تر بررسی شد، می­توانیم از چندین سرویس‌دهنده به‌طور هم‌زمان استفاده کنیم تا بتوانیم حجم بسته‌های داخل صف را مدیریت کنیم. در این سیستم یک پارامتری تحت عنوان پیش­دستی[[19]](#footnote-19) تعریف می­شود، به این معنا که تعداد بسته‌های درخواستی از صف را کنترل می­کند که بسته‌ها به‌صورت چندتایی ارسال شوند. در این حالت، بسته‌ها به حالت گذرا تغییر وضعیت یافته تا هنگامی‌که سرویس­دهی این بسته‌ها تمام شد، یک تعداد بسته جدید تقاضا شود. این امر موجب می­شود تعداد پیام‌هایی که برای ارسال و دریافت بین سرور صف و سرویس‌دهنده منتقل می­شود، کاهش یابد و درنتیجه بازدهی بالاتر رود. برای مثال به شکل ‏4–13 توجه کنید.



**شکل ‏4–13-اتصال دو مصرف‌کننده‌ی اطلاعات به یک صف**

فرض کنید در این شکل دو مصرف‌کننده‌ی اطلاعات، به این صف متصل شده‌اند و در صف 50 بسته وجود دارد. C1 با پیش­دستی 5 به این صف متصل شده و C2 که کامپیوتری با امکانات سخت‌افزاری قوی­تری است، با پیش­دستی 10 به این صف متصل می­شود. حال در لحظه صفر C1 درخواست 5 بسته و C2 درخواست 10 بسته می­کند. در این حالت همچنان 50 بسته در صف وجود دارد، با این تفاوت که 15 بسته به حالت گذرا درآمده است. بعد از پردازش توسط C1 وC2 اعلان پایان کار داده و تقاضای بسته جدید کرده به این شکل بسته‌ها از صف حذف‌شده و بسته‌های جدید برای آن‌ها ارسال می­شود.

### راه­اندازی RabbitMQ با داکر

از پیش­تر در بخش مروری بر داکر به یاد داریم که می­توان ایمیج های استاندارد را از داکرهاب، بارگذاری و مورداستفاده قرارداد. RabbitMQ خود یکی از ایمیج های استاندارد است که به‌سادگی از آن استفاده می­شود. برای این کار می­توانیم در داخل داکر فایل از ایمیج استاندارد شروع کرده و پلاگین­هایی که قصد فعال‌سازی آن را داریم، با اجرای دستور مربوطه به‌کارگیریم تا ایمیج جدیدی با ویژگی‌هایی که مدنظر داریم ساخته شود. در شکل ‏4–14 محتوای داکر فایل را می‌توان مشاهده کرد.

FROM rabbitmq:3-management

RUN rabbitmq-plugins enable rabbitmq\_management

RUN rabbitmq-plugins enable  rabbitmq\_mqtt –offline

شکل ‏4–14-محتوای داکر فایل برای راه‌اندازی RabbitMQ

حال با به­کارگیری داکر کامپوز تنها کافی است دستور دهیم که ایمیج فوق ساخته شود و پورت مخصوص به ارتباط با کانتینر و کامپیوتر میزبان برقرار شود. همچنین می­توان فایل تنظیمات را تغییر داد و آن را در داخل کانتینر قرار داد. در شکل ‏4–15 می­توان دستوراتی که در فایل docker-compose.yaml برای بالا آمدن کانتینر RabbitMQ نوشته‌شده را مشاهده کرد.

  queue:

    build: rbmq/.

    container\_name: rabbitmq

    ports:

      - 5672:5672

      - 15672:15672

      - 1884:1884

    volumes:

      - "./rbmq/Conf/:/etc/rabbitmq/"

شکل ‏4–15- بخش مرتبط با RabbitMQ در داکر کامپوز

## مروری بر ذخیره کننده Redis

Redis یک ذخیره کننده ساختار اطلاعات، داخل حافظه‌ای متن‌باز است که می­تواند به‌عنوان پایگاه­داده، حافظه نهان و ابزاری برای یکپارچه‌سازی عمل کند. Redis از ساختارهای اطلاعات متفاوتی ازجمله؛ رشته­ها، هش­ها، لیست­ها، مجموعه­ها، مجموعه­های مرتب‌شده برای کاربردهای متفاوت و جریان­های اطلاعات (سری زمانی‌ها) پشتیبانی می­کند و توانایی اجرا بر روی خوشه­ای از کامپیوترها را دارا است که بزرگ‌ترین مزیت استفاده از Redis است.

در این نرم‌افزار بعضی از امکانات محاسباتی ازجمله اضافه کردن به رشته‌ها، تغییر در اعداد، تغییر در لیست‌ها، محاسبه اشتراکات دو مجموعه، بررسی المان‌های منحصربه‌فرد و ...را انجام داد. Redis به زبان **ANSI C** نوشته‌شده و بر روی همه‌ی سیستم‌عامل‌ها بدون هیچ‌گونه پیش‌نیاز قابل‌اجرا است. اگرچه این برنامه بر روی همه‌ی سیستم‌عامل‌ها توانایی اجرا دارد ولی خود برنامه، استفاده از سری سیستم‌عامل‌های Linux را پیشنهاد می­دهد.

درست است که Redis اطلاعات را در حافظه ذخیره می‌کند اما به این معنی نیست که پس از خاموش شدن و یا هر اتفاقی که باعث خالی شدن حافظه شود، داده‌های ما پاک می‌شوند. بلکه Redis برای نگه‌داری دائمی داده‌ها آن‌ها را با توجه به تنظیماتی که برای آن مشخص کرده‌ایم به دیسک اصلی سیستم منتقل کرده و بعد از پاک شدن حافظه مجدد می‌تواند آن‌ها را منتقل کند و کار را از سر بگیرد. این ویژگی باعث شده اصطلاحاً به آن **on-disk persistence** بگویند.

زمانی از ذخیره‌سازی موقت[[20]](#footnote-20) استفاده می‌شود که قصد داشته باشیم دسترسی به هارددیسک کمتر انجام شود. به‌عبارت‌دیگر در ذخیره­سازی موقت، اطلاعات در حافظه موقت ذخیره می‌شود که این فرآیند سرعت دسترسی به اطلاعات و بارگذاری آن‌ها را افزایش می‌دهد. از این طریق در کنار صرفه‌جویی در زمان و افزایش سرعت، دسترسی کمتری به منابع موردنیاز انجام می‌شود. این امر نیز به بهینه‌سازی بیشتر کمک می‌کند. به این نکته نیز باید اشاره کرد که در Redis اطلاعات در حافظه ذخیره می‌شوند، این امر باعث می‌شود دسترسی به آن‌ها با سرعت بسیار بیشتری انجام شود؛ اما این سکه روی دیگری نیز دارد و امکان ذخیره‌سازی دائمی اطلاعات را در Redis وجود نخواهد داشت. در این پژوهش ازآنجاکه پس از تجمیع اطلاعات نیازمند پردازش این اطلاعات هستیم تا به رانندگی افراد امتیازی اختصاص دهیم از این نرم‌افزار استفاده شد.

در Redis دو عنصر، کلید و مقدار[[21]](#footnote-21) داریم. عنصر مقدار می­تواند انواع مختلفی داشته باشد. اجازه دهید به برخی این مقادیر و کاربردهای آن‌ها نگاهی بیندازیم.

**رشته‌ها:** اگر مقدار از نوع رشته بود، می­توان عملیات درج، بهنگام سازی، حذف، دریافت را از آن کلید انجام داد. مثلاً با دستور زیر می­توان یک کلید ساخت و یک‌رشته در آن درج کرد:

|  |  |
| --- | --- |
| یک کلید به اسم user\_1 ساخته‌شده و  book1 به‌عنوان مقدار برای این کلید در نظر گرفته می­شود. | SET user\_1 book1 |
| مقدار مورد (book1) نظر برای کلید user\_1 را برگردانده می­شود. | GET user\_1 |

**لیست:** با استفاده از لیست می­توان یک آرایه دلخواه داشت که بتوان در این آرایه، عنصری را اضافه یا کم کرد و عملیات مختلف دیگر را انجام داد.

**دسته‌ها**[[22]](#footnote-22)**:** اگر مقدار موجود از نوع دسته‌ها باشد درواقع یک لیست وجود دارد که هیچ‌کدام از عناصر آن تکراری نیستند.

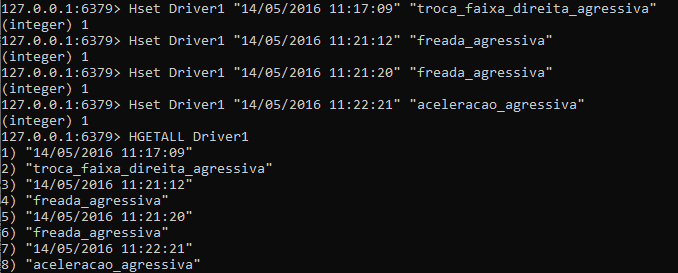
**دسته‌های منظم شده**[[23]](#footnote-23)**:** همانند مقدارهایی از نوع دسته است با این تفاوت که هر عنصر از مجموعه دارای وزن است و این وزن می­تواند به‌صورت مرتب نگهداری شود. (مثلاً از وزن کم به زیاد)

**هش­ها[[24]](#footnote-24):** اگر با انواع JSON آشنایی داشته باشید درک مقادیر هش‌ها ساده­تر است. این مقادیر می­توانند اشیایی مانند JSON را در خود جای دهند. بدین شکل که ابتدا کلیدی برای ذخیره‌سازی اطلاعات دریافت کرده و سپس به ذخیره‌سازی اطلاعات به‌صورت زوج مرتب‌هایی از کلید و مقدار می­پردازد.

علاوه بر موارد فوق، Redis می­تواند مقادیر دیگری را نیز ذخیره و بازیابی کند. یکی از انواع آن‌ها، HyperLogLog است. فرض کنید می­خواهید تعداد تکرار یک عنصر خاص از یک لیست را به دست آورید. اگر این لیست بسیار بزرگ باشد، این کار به‌راحتی انجام نمی­پذیرد. مقادیر HyperLogLog می‌تواند با دقت بسیار بالا (اما نه ۱۰۰ درصد) تعداد تکرار یک عنصر خاص را حدس بزند. این کار با استفاده از الگوریتم‌های خاصی امکان‌پذیر است. انواع دیگری مانند ذخیره‌سازی عناصری از جنس موقعیت‌های مکانی و یا Bitmap های نیز در Redis وجود دارند که کاربردهای خاص خود را دارند.

در این پژوهش ازآنجاکه میان رانندگان و اطلاعات ارسالی آن‌ها تمایز قائل هستیم و تنها قصد انتقال رویدادهای رانندگی و برچسب زمانی آن‌ها را داریم، می­توانیم اطلاعات خود را به‌صورت هش‌ها در Redis درآورده و به این شکل ذخیره‌سازی اطلاعات را پیاده­سازی کنیم. اطلاعات با دستور و ساختار زیر ذخیره‌شده‌اند.

کلید ذخیره‌سازی اطلاعات بر اساس شماره شناسایی راننده تعیین می­شود. ازآنجایی‌که رشته­های هر کلید باید نسبت به یکدیگر منحصربه‌فرد باشند، از برچسب زمان به‌عنوان رشته و در بخش مقدار آن رویداد رانندگی موردنظر نوشته می­شود. با دستور HSET می­توان یک سری اطلاعات هش ساخت و اطلاعات رشته و مقدار را به آن، مانند لیست، اضافه کرد و با دستور HGETALL می­توان اطلاعات ذخیره‌شده در یک هش را به دست آورد. در شکل ‏4–16 نمونه­ای از استفاده از این دستورات و بارگذاری این اطلاعات در سرویس‌دهنده Redis می­توانیم مشاهده کنیم.



شکل ‏4–16- بررسی دستورات کار کردن با هش­ها در Redis

در شکل فوق با برنامه redis-cli به سرور Redis متصل شدیم و سعی کردیم دستورات را در خط دستور وارد کنیم. در ادامه با یک برنامه پایتون، سعی می­شود اطلاعات را از نرم‌افزار RabbitMQ خوانده و به سرویس‌دهنده Redis منتقل کرد.

### راه­اندازی Redis با داکر کامپوز

از پیش‌تر در بخش مروری بر داکر به یاد داریم که می­توان ایمیج های استاندارد را از داکرهاب، بارگذاری و مورداستفاده قرارداد، Redis یکی از ایمیج های استاندارد است که می­توان به‌سادگی از آن استفاده نمود، برای استفاده از این ایمیج با داکر کامپوز، تنها کافی است، پورت مخصوص به ارتباط با کانتینر آن را به کامپیوتر میزبان متصل کرده و فضایی در کامپیوتر میزبان برای ذخیره‌سازی اطلاعات برای آن اختصاص دهیم تا با شروع به کار مجدد کانتینر، اطلاعات ذخیره‌شده در آن از بین نرود، همچنین می­توان فایل تنظیمات مربوط به Redis را تغییر داد و در کانتینر قرارداد و دستور داد تا برنامه Redis بر اساس تنظیماتی که در کانتینر قرارگرفته، اجرا شود. در شکل ‏4–17 می­توان دستوراتی که در فایل docker-compose.yaml برای بالا آمدن کانتینر Redis نوشته‌شده را مشاهده کرد.

cache:

    image: redis:latest

    container\_name: redis

    ports:

        - 6379:6379

    volumes:

        - ./redis/config/redis.conf:/redis.conf

        - ./redis/Data:/data

    command: [ “redis-server”, “/redis.conf” ]

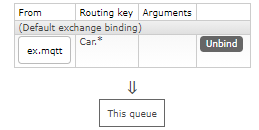
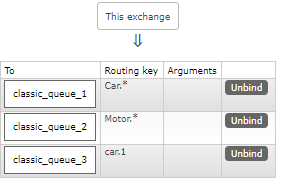
شکل ‏4–17- بخش مرتبط با Redis در داکر کامپوز

## پیاده‌سازی سیستم دریافت و ذخیره‌سازی

در این بخش قصد داریم تا به بررسی کلی سیستم دریافت و ذخیره‌سازی اطلاعات بپردازیم، در بخش­های قبل با نرم‌افزارهای مورداستفاده آشنا شدیم و راه­اندازی آن‌ها با کمک داکر را بررسی کردیم. حال به بررسی عملکرد هرکدام از نرم‌افزارهای فوق در معماری مورداستفاده به‌صورت جدا پرداخته و ارتباط آن‌ها با یکدیگر را معرفی می­کنیم.

همان‌طور که پیش­تر در شکل ‏4–8 مشاهده کردیم، برای پیاده‌سازی این سیستم از سه برنامه مجزا استفاده شد که با یکدیگر در تعامل هستند. برنامه‌هایی به شرح: صف[[25]](#footnote-25) مصرف­کننده[[26]](#footnote-26) ذخیره موقت[[27]](#footnote-27).

برنامه صف با کمک RabbitMQ پیاده‌سازی شد. بدین شکل که خودروها با پروتکل MQTT بر روی پورت 1884 با شناسه هویتی مشخص، به این نرم‌افزار متصل و بسته‌های خود را با تاپیک مشخص (شماره خودروی خود) به مرکز انتقال "ex.mqtt" منتقل می­کنند. در ادامه همان‌طور که در شکل ‏4–18 می­بینید، بسته‌ها با کلیدهای مسیریابی متفاوت وارد صف‌های مربوط به خود می­شوند. برای مثال بسته‌ای با تاپیک Car.321 بر اساس کلیدمسیریابی Car.\* به‌صف classic\_queue\_1 منتقل می­شود و در صف نگهداری می­شود تا یک مصرف‌کننده اطلاعات را از صف بخواند.



شکل ‏4–18- فرایند انتقال بسته‌ها از بخش توزیع دادگان به صف­ها

در ادامه می­بایست نرم‌افزاری طراحی کنیم که اطلاعات را از صف‌های RabbitMQ خوانده و به Redis منتقل کند. تا فرایند بعدی که امتیاز دادن به رانندگی شخص در هر سفر است، انجام شود.

می­دانیم می­توان بسته‌های قرارگرفته داخل صف را بر روی پورت 5672 و پروتکل AMQP-0-91 با شناسه هویتی مشخص توسط کتابخانه pika که در پایتون نوشته‌شده، دریافت کنیم. برای انجام این کار به‌صورت بهینه باید حجم بسته‌های درون صف RabbitMQ از حدی بیشتر نشود. مکانیزم دریافت اطلاعات از صف به‌صورت موازی و چند رشته پیاده­سازی شد. بدین شکل که با کمک مکانیزم برنامه‌نویسی چند رشته‌ای و کتابخانه threading، می­توان به تعداد دلخواه، مصرف­کننده اطلاعات ساخت که به صف­ها متصل شوند. کلاس مربوط به این مدل را می­توان در شکل ‏4–19 مشاهده کرد.

class rbmq(threading.Thread):

    # Thread class with a \_stop() method.

    def \_\_init\_\_(self,Slug,Parameter,prefetch\_count,QueueName,CalbackFunc,\*args, \*\*kwargs):

        super(rbmq, self).\_\_init\_\_(\*args, \*\*kwargs)

        self.slug=Slug

        self.parameters=Parameter

        self.pckh=prefetch\_count

        self.queuename=QueueName

        self.callbackfunc=CalbackFunc

    def run(self):

        # target function of the thread class

        try:

            connection = pika.BlockingConnection(self.parameters)

            channel = connection.channel()

            channel.basic\_qos(prefetch\_count=self.pckh)

            channel.basic\_consume(queue=self.queuename,

                          on\_message\_callback=self.callbackfunc,

                          consumer\_tag=self.slug)

            print( self.slug ,' [\*] Waiting for messages on :',self.queuename )

            channel.start\_consuming()

        finally:

            connection.close()

            print(self.slug ,'  Stoped from ',self.queuename )

    def get\_id(self):

        # returns id of the respective thread

        if hasattr(self, '\_thread\_id'):

            return self.\_thread\_id

        for id, thread in threading.\_active.items():

            if thread is self:

                return id

    def stop(self):

        thread\_id = self.get\_id()

        res = ctypes.pythonapi.PyThreadState\_SetAsyncExc(thread\_id,

              ctypes.py\_object(SystemExit))

        if res > 1:

            ctypes.pythonapi.PyThreadState\_SetAsyncExc(thread\_id, 0)

            print('Exception raise failure')

شکل ‏4–19-کلاس مربوطه به برنامه انتقال‌دهنده اطلاعات RabbitMQ به Redis

در ادامه وظیفه‌ای که هرکدام از این رشته‌ها دارند، هنگام دریافت پیام تعیین می­شود که با پیام دریافت شده، چه عملی را انجام دهند. برای انجام این کار هنگامی‌که قصد داریم شئ ای از کلاس فوق بسازیم تابعی را به‌عنوان CallbackFunc به آن معرفی می­کنیم تا وظایفی که هنگام دریافت بسته­ها باید انجام شود را تعریف کنیم. هنگامی‌که بسته‌ها از صف‌ها دریافت شدند، باید به نرم‌افزار ذخیره­سازی موقت (Redis) منتقل شوند. در بخش قبل بررسی کردیم که برای این کار از مکانیزم هش‌ها در Redis استفاده خواهیم کرد. به همین منظور با استفاده از کتابخانه Redis در پایتون، به‌صورت کاربر Redis به این سرویس­دهنده متصل شده و اطلاعات خارج‌شده از صف را در این نرم‌افزار قرار می­دهیم.

در شکل ‏4–20 تابعی که وظیفه انتقال اطلاعات به سرویس‌دهنده Redis را دارد مشاهده می­کنیم، درصورتی‌که این برنامه نتواند اطلاعات را به Redis منتقل کند، اطلاعات را در بخشی از کانتینر اجراکننده این برنامه که به کامپیوتر میزبان متصل شده، منتقل می­کند. اگر روند اجرای برنامه مشکلی نداشت، پیام تصدیق برای برنامه صف ارسال می­شود تا پیام را از حافظه خود پاک کند.

def Packet\_Handeler\_callback(ch, method, properties, body):

    Str=str(body.decode("utf-8"))

    Data=json.loads(Str)

    client = redis.Redis(host='redis', port=6379)

    send\_correct = client.hset(name=Data['driver\_id'],key=Data['timestamp'],value=Data['event'])

    # if Fails , save on log file

    if not(send\_correct):

        with open(".//log//log.txt", "a") as myfile :

            myfile.write(Str)

    ch.basic\_ack(delivery\_tag = method.delivery\_tag)

شکل ‏4–20-تابع فراخوانی اطلاعات هنگام دریافت اطلاعات از Redis و انتقال به RabbitMQ

## جمع بندی

در بخش‌های فوق هر یک از نرم‌افزارهای به‌کاررفته برای دریافت و ذخیره­سازی اطلاعات بررسی کردیم و در پایان برنامه‌ای که وظیفه یکپارچه­سازی این نرم‌افزارها را دارد، مرور کردیم، حال وقت آن رسیده که به بررسی اطلاعاتی که در سرویس‌دهنده Redis جمع شده بپردازیم. تا رانندگی افراد را مورد ارزیابی قرار دهیم.

# فصل سوم طراحی و معماری برنامه های کاربردی

**طراحی و معماری**

## گوجه فروشی حسین

شسشس

## خیار فروشی محمد

شسکخیدشس

## جمع بندی

کمسبیتدسمنتب

# فصل چهارم بکارگیری و نحوه کارکرد سیستم برای کاربران

**بکارگیری سیستم**

پیاده‌سازی کرده تا اتفاقی که هنگام عملی شدن پروژه رخ می­دهد را شبیه‌سازی کنیم.

## راه اندازی و پیشنیاز ها

سسشیی

## جمع‌بندی فصل چهارم

در این بخش به بررسی و آزمودن بخش‌های مختلف پروژه پرداختیم، ابتدا اهمیت هرکدام از حسگرها را با بررسی فرایند آموزش بر روی اطلاعات تنها یک حسگر انجام دادیم و دریافتیم که حسگر شتابسنج اهمیت بیشتری دارد. سپس با انجام آزمایشی دیگر بر روی اطلاعات هر دو حسگر، فرآیند آموزش را بدون به‌کارگیری پنجره‌بندی اتفاقی انجام دادیم و اثر آن را در تشخیص هر رخداد مشاهده نمودیم. در پایان نیز با کمک یک برنامه شبیه‌سازی، نحوه عملکرد سیستم دریافت و ذخیره‌سازی اطلاعات را در ابرمحاسباتی بررسی کردیم؛ و بسته‌هایی که یک راننده نمونه به این سیستم ارسال می­کند را ارزیابی کردیم.

# فصل پنجم نتیجه‌گیری و پیشنهادها



## نتیجه­گیری

در این رساله ما یک سیستم کامل برای پیاده‌سازی در شرکت‌های بیمه‌ای، استفاده در پلیس راهنمایی و رانندگی و شرکت­های حمل‌ونقل ارائه کردیم که با به‌کارگیری آن بتوان میزان ایمن بودن رانندگی افراد را سنجید و در یک شرکت بیمه‌ای، برای تعیین حق بیمه، در یک سیستم عادلانه متناسب با نوع رانندگی افراد برای آن‌ها مبلغ، تعیین نمود تا بتوان رانندگان را با انجام این روش به رانندگی ایمن تشویق کرده تا جاده‌ها و خیابان‌هایی ایمن‌تری ایجاد شود و آمار تصادفات شهری و تلفات جاده‌ای را کاهش یابد. برای این کار ابتدا مدلی با به‌کارگیری حسگرهای شتابسنج و ژیروسکوپ طراحی شد که میزان رخدادهای خطرناک رانندگی با دقتی قابل‌قبول تشخیص دهند. رخدادهایی همچون گردش‌به‌راست خطرناک باسقت 91 درصد (87 درصد[[28]](#footnote-28))، گردش‌به‌چپ خطرناک با دقت 95 درصد (83 درصد)، شتابگیری خطرناک با دقت 78 درصد (69 درصد)، ترمزگیری خطرناک با دقت 91 درصد (73 درصد)، تعویض لاین خطرناک به سمت راست با دقت 84 درصد (69 درصد)، تعویض لاین خطرناک به سمت چپ با دقت 83 درصد (73 درصد) به‌دست‌آمده است. سپس یک سیستم دریافت و ذخیره‌سازی اطلاعات طراحی شد که بتواند بسته‌های تولیدی توسط خودروها را در یک سامانه ابری ذخیره‌سازی نماید. برای این کار از نرم‌افزار RabbitMQ به‌عنوان مبادله گر اطلاعات و Redis به‌عنوان پایگاه داده موقت استفاده شد، همچنین برنامه با زبان پایتون نوشته شد که اطلاعات را از صف‌های RabbitMQ خوانده و به Redis منتقل کند. کل این مجموعه بر بستر کانتینرهای داکر طراحی شد تا توانایی انتقال به هر ارائه‌دهنده سرویس ابری را داشته باشد. در پایان نیز یک مدل مبتنی بر اطلاعات آماری طراحی شد که به رانندگان بر اساس زمان رانندگی و تعداد رویدادهای خطرناک تولیدشده امتیازی اطلاق کند. تا بتوان رانندگان را با یکدیگر بر اساس این امتیاز مقایسه کرد تا رانندگان ایمن را از رانندگان خطرناک متمایز کرده و حق بیمه‌ای متناسب با نوع رانندگی آن‌ها در نظر گرفت.

## پیشنهادها

در این بخش پژوهش‌هایی را که می­توان در ادامه راه این پژوهش انجام شود ذکرشده:

1-بهینه‌سازی با دقت بالاتر پارامترهای پنجره‌بندی اتفاقی برای تشخیص رخدادهای خطرناک

در این پروژه برای تشخیص رویدادهای خطرناک، از پنجره­بندی اتفاقی استفاده شد که با کمک پارامترهای آن، چندین مدل آموزش داده‌شده‌اند و مدلی که دقت بیشتری بر روی دادگان تست در اختیار ما قرار می­داد، به‌عنوان بهترین مدل انتخاب شد، ولی شیوه‌ای که برای تغییر این پارامترها بکار گرفته شد، برای پارامتر احتمال نگاه رو به عقب بافاصله‌ی 10 درصد و برای پارامتر میانگین حرکت پنجره روبه‌جلو عدد 5 درصد در نظر گرفته شد. به‌عنوان کارهای آینده می­توان گستره بررسی را افزایش داد تا مدل‌های بیشتری موردبررسی قرار گیرند و نتایج بهتری به‌دست­آید.

2-استفاده از مجموعه دادگان دیگری برای امتیازدهی رانندگان بر اساس میزان اهمیت رخداد خطرناک

در فرایند نمره­دهی به رانندگان، هنگامی‌که میزان احتمال خطرناک بودن رخدادها را محاسبه کردیم، این اعداد را به‌صورت مستقیم با یکدیگر جمع نمودیم و میزان خطرناک بودن، رخداد ترمزگیری خطرناک و گردش‌به‌چپ خطرناک را یکسان در نظر گرفتیم. درحالی‌که می­توان با کمک مجموعه دادگانی دیگر در زمینه تصادفات میزان اهمیت هر یک از رخداد را به‌صورت دقیق اندازه گرفت و در فرایند نمره دهی به رانندگان بهره برد.

3-استفاده از اطلاعات GPS و تشخیص رویدادهایی مربوط به‌سرعت غیرمجاز

برای فرایند تشخیص رخدادهای خطرناک، می­توان با اضافه کردن حسگر GPS سرعت حرکت خودرو را محاسبه کرد و رخدادهای مربوط به‌سرعت غیرمجاز در محل را اندازه‌گیری نمود. همچنین با کمک این اطلاعات می­توان محل‌های پرخطر را تشخیص داد.

4-طراحی یک داشبورد برای نمایش اطلاعات و ارزیابی عملکرد رانندگان

در این پروژه داشبوردی برای نمایش اطلاعات پردازش‌شده، طراحی نشد، ولی یکی از بخش­هایی که این پروژه نیاز دارد تا به مرحله عمل برسد، طراحی و پیاده‌سازی یک سرویس‌دهنده تحت وب است که اطلاعات پردازش‌شده و امتیازی که رانندگان کسب نموده‌اند را به آن­ها اطلاع‌رسانی کند.

# مراجع

|  |  |
| --- | --- |
| [1] | https://www.eghtesadnews.com/fa/tiny/news-273274, تعداد خودرو ها در سطح شهر, تهران: اقتصاد نیوز, 1397. |
| [2] | Docker overview, https://docs.docker.com/, 2021. |
| [3] | https://www.rabbitmq.com/documentation.html, RabbitMQ documentation 3.8.4, 2021. |

**Abstract**

Dangerous driving is one of the most important causes of car accidents. If drivers know that their behavior is being recorded while driving, they will be driving safer. Also, if it is possible to implement a system that encourages people to drive safely by being aware of the type of driving behaviors of individuals, this will lead to a reduction in traffic accidents. This project aims to implement a method for detecting driving events (including dangerous right and left lanes, dangerous lane changes left and right, dangerous braking, and acceleration) with the help of sensor information so that driver information Receive and store in a supercomputer and use a solution to calculate how safe people are driving. More precisely, using a new windowing method on time series, a decision tree on window similarities (using the Fast-DTW algorithm) was trained to detect driving events, then to receive and Data storage RabbitMQ was used as a message exchanger, and Redis was used as a temporary database. Finally, a statistical method was used to score and compare people's driving. The dangerousness of driving introduces them as a benchmark for comparison. With the help of the results presented in this study, an insurance plan can be implemented so that drivers who show safer driving pay fewer insurance premiums so that this plan can be used as an incentive for Improved driving culture was considered.

**Key Words:** Internet of Things, Machine learning, Driver Profiling, Driving culture

1. id3 with prune [↑](#footnote-ref-1)
2. integration [↑](#footnote-ref-2)
3. Message Broker [↑](#footnote-ref-3)
4. Docker [↑](#footnote-ref-4)
5. container [↑](#footnote-ref-5)
6. Daemon process [↑](#footnote-ref-6)
7. Command line [↑](#footnote-ref-7)
8. CI(continuous integration) [↑](#footnote-ref-8)
9. CD(continuous delivery) [↑](#footnote-ref-9)
10. Portable [↑](#footnote-ref-10)
11. Producer [↑](#footnote-ref-11)
12. Consumer [↑](#footnote-ref-12)
13. Queue [↑](#footnote-ref-13)
14. Router(Exchange) [↑](#footnote-ref-14)
15. Channels [↑](#footnote-ref-15)
16. Routing key [↑](#footnote-ref-16)
17. Transient [↑](#footnote-ref-17)
18. Persistent [↑](#footnote-ref-18)
19. Prefetch [↑](#footnote-ref-19)
20. Caching [↑](#footnote-ref-20)
21. Value [↑](#footnote-ref-21)
22. Sets [↑](#footnote-ref-22)
23. Sorted Sets [↑](#footnote-ref-23)
24. Hashes [↑](#footnote-ref-24)
25. Queue [↑](#footnote-ref-25)
26. Consumer [↑](#footnote-ref-26)
27. Cache [↑](#footnote-ref-27)
28. اعداد دقت اعلام شده دقت precision و اعداد داخل پردانتز معیار دقت f1-score است. [↑](#footnote-ref-28)