

دانشگاه صنعتي امیرکبیر  
(پلی تکنیک تهران)

دانشكده مهندسی کامپیوتر و فناوری اطلاعات

بررسی و مرور کلی پروژه کارشناسی

گرایش فناوری اطلاعات

پیاده‌سازی ابزار مبتنی وب به منظور سجنش کارایی رابط کاربری سامانه‌های مبتنی بر وب به روش جمع‌سپاری

نگارش

امیر حقیقتی ملکی

استاد راهنما

استاد احمد عبداله‌زاده بارفروش

تیر ۱۳۹۷

اينجانب امیر حقیقتی ملکی متعهد مي‌شوم كه مطالب مندرج در اين پايان نامه حاصل كار پژوهشي اينجانب تحت نظارت و راهنمايي اساتيد دانشگاه صنعتي اميركبير بوده و به دستاوردهاي ديگران كه در اين پژوهش از آنها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذكر گرديده است. اين پایان نامه قبلاً براي احراز هيچ مدرك هم‌سطح يا بالاتر ارائه نگرديده است.

در صورت اثبات تخلف در هر زمان، مدرك تحصيلي صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پيگيري قانوني خواهد داشت.

كليه نتايج و حقوق حاصل از اين پایان نامه متعلق به دانشگاه صنعتي اميركبير مي‌باشد. هرگونه استفاده از نتايج علمي و عملي، واگذاري اطلاعات به ديگران يا چاپ و تكثير، نسخه‌برداري، ترجمه و اقتباس از اين پایان نامه بدون موافقت كتبي دانشگاه صنعتي اميركبير ممنوع است.   
نقل مطالب با ذكر مآخذ بلامانع است.

امیر حقیقتی ملکی

امضا

تقدیر و تشکر

از زحمات استاد احمد عبداله‌زاده بارفروش و همچنین آقای دکتر میثم نظریانی که بدون همکاریشان به ثمر رسیدن این پروژه ممکن نبود نهایت تشکر و قدردانی را دارم.

چكيده

با تقریب خوبی می‌توان گفت تمامی مدل‌های کیفی نرم‌افزار، کارایی را جزو مشخصه‌های اصلی کیفیت یک نرم‌افزار مطرح می‌کنند. وجه مشترک تعاریف متعددی که برای کارایی مطرح می‌شود، در سه بعد کاربر، انجام یک فعالیت مشخص و تعامل با یک واسط برای انجام آن فعالیت، قابل بیان است. به عنوان یک مهندس نرم‌افزار، افزایش کیفیت در محصولات و کاهش هزینه‌های ناشی از خرابی‌ها و یا درخواست‌های تغییر، چالشی تامل برانگیز است. وب‌اپلیکیشن‌ها به عنوان نوعی محصول نرم‌افزاری که در آن‌ها زیبایی، واسط کاربری و نحوه تعامل کاربران مهم است، به دلیل استفاده گسترده‌شان، می‌توانند تاثیر شگرفی در موفقیت یک پروژه صنعتی، کسب‌وکارهای نوپا و یا تسهیل زندگی روزمره با استفاده از نرم‌افزارها داشته باشند. از جمله نقاط ضعف بیشتر وب‌اپلیکیشن‌ها، طراحی نه‌چندان کاربرپسندانه واسط کاربری آن‌هاست که موجب شده تا در بسیاری از موارد، کاربران، علاقه‌مندی استفاده از محصول مبتنی وب یک سازمان را در عین سرمایه‌گذاری‌های زیاد آن سازمان برای جذب کاربر، از دست بدهند و در نتیجه متضرر شوند. گرچه، به صورت ایده‌آل، تمامی تصمیم‌گیری‌های مدیریتی و کلان (از قبیل اتخاذ مدل‌های فرایندی مناسب برای تولید نرم‌افزار با هزینه کم) با نهایت دقت و تجربه انجام می‌شوند، ولی در بسیاری از موارد همچون پروژه تقویم شرکت گوگل، مواردی ملاحظه می‌شود که واسط کاربری ناکارآمد، به ناچار، هزینه‌های گاهاً زیادی به تیم مهندسی نرم‌افزار تحمیل کرده است. با مروری بر منابع مختلف، ارزیابی و تست روی نمونه‌های اولیه رابط کاربری وب‌اپلیکیشن‌ها به منظور رفع نواقص آن‌ها، امری واضح به نظر می‌رسد. اما پاسخ دادن به این سوال که «چه واسط کاربری‌ای خوب است؟» همیشه آسان نبوده و با تغییر فناوری و گذشت زمان شاهد تغییر سریع در نیازمندی‌ها هستیم که شاید چک‌لیست‌ها و توصیه‌ها نیز پاسخگوی دقیقی برای آن‌ها نباشند. بنابراین می‌بایست در طراحی واسط کاربری، به یک روش کمی و قابل استناد، نیازمندی‌ها را با استفاده از نمونه‌های اولیه بسنجیم (که به دلیل هنری انگاشتن اکثر کارها، این امر نادیده گرفته می‌شود). اما سنجش دقیق، نیازمند جمع‌آوری داده از ارزیابی و تست واسط کاربری توسط کاربران نهایی است تا بتوان تحلیل دقیق انجام داد و مشکلات طراحی واسط را به درستی تشخیص داد. یکی از روش‌های جمع‌آوری داده، استفاده از جمع‌سپاری است. باید توجه داشت که استفاده از جمع‌سپاری چالش‌هایی را فرارویمان خواهد گذاشت که از جمله آن‌ها می‌توان به عدم وجود صحت در داده‌ها اشاره کرد. در این پروژه وب‌اپلیکیشنی به منظور ارائه داشبورد مدیریتی برای صاحبان طراحی و افراد متمایل به انجام تست‌های مختلف با معیارهای متفاوت و دلخواه، پیاده خواهد شد. همچنین دادگان و پاسخ‌ها و تحلیل‌های تست اپلیکیشن در مواجهه کاربران واقعی با آن‌ها، به اطلاع کاربر خواهد رسید؛ علاوه بر موارد فوق، قسمت اصلی این پروژه در پاسخ به چالش صحت داده در روش جمع‌سپاری، ابتدا رفتار کاربران پاسخ‌دهنده (کارگران) توسط ماتریسی مدل می‌شود که برای مدل‌سازی و به دست آوردن مقادیر مدل‌ها، از روش تزریق سوالات طلایی استفاده خواهد شد. سپس در صورت پایین بودن کیفیت کار کارگران از حد مشخصی که در هنگام مدل‌سازی مشخص می‌شود، نتیجه کار آن‌ها به عنوان داده نامربوط شناخته شده و حذف می‌گردد. امکان تعریف تست‌های دلخواه و محدود نبودن به تست‌های از پیش تعریف شده تفاوت عمده ابزار کارا با سایر ابزارهای مشابه است؛ از جمله ابزارهای مطرح موفق در این حوزه، می‌توان به UsabilityHub، Optimizely و CrazyEgg اشاره کرد که همانطور که ذکر شد، در طی این پروژه، سعی بر برطرف‌سازی برخی از نواقص آن‌هاست.

واژه‌های کلیدی:

ویرایش شود

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

[۱ فصل اول مقدمه 1](#_Toc518837713)

[مقدمه 2](#_Toc518837714)

[1-1- مقدمه‌ای بر رایانش ابری 3](#_Toc518837715)

[1-1-1- تاریخچه رایانش ابری 3](#_Toc518837716)

[1-1-2- مدل‌های سرویس در رایانش ابری 4](#_Toc518837717)

[1-2- مقدمه‌ای بر کانتینرها 7](#_Toc518837718)

[1-3- داکر چیست؟ 8](#_Toc518837719)

[1-4- هماهنگ‌کننده‌ی کانتینری چیست؟ 9](#_Toc518837720)

[۱-۵- تعریف مساله 10](#_Toc518837721)

[۲ فصل دوم مفاهیم اولیه 11](#_Toc518837722)

[مفاهیم اولیه 12](#_Toc518837725)

[۲-۱- رویکردهای توسعه‌پذیری 12](#_Toc518837726)

[2-1-1- توسعه‌پذیری خودکار 12](#_Toc518837727)

[2-1-2- توسعه‌پذیری زمان‌بندی شده 13](#_Toc518837728)

[2-1-3- توسعه‌پذیری پیش‌بینی کننده 13](#_Toc518837729)

[2-2- هماهنگ‌کننده‌های کانتینری و ویژگی‌های آن 14](#_Toc518837730)

[2-2-1- هماهنگ‌کننده کانتینری داکر سوارم 14](#_Toc518837731)

[2-2-2- هماهنگ‌کننده‌ی کانتینری کوبرنتیز 16](#_Toc518837732)

[۳ فصل سوم طراحی و پیاده‌سازی 18](#_Toc518837733)

[طراحی و پیاده‌سازی 19](#_Toc518837735)

[3-1- شرح کلی طراحی 19](#_Toc518837736)

[3-2- نمودارهای تحلیل 21](#_Toc518837737)

[3-2-1- نمودار یوزکیس 21](#_Toc518837738)

[۳-۲-۲- نمودار کانتکست 22](#_Toc518837739)

[۳-۳- متدلوژوی مورد استفاده 23](#_Toc518837740)

[۳-۳-۱- الگوی آبشاری 23](#_Toc518837741)

[3-4- ابزار مورد استفاده 26](#_Toc518837742)

[۴ فصل چهارم تکنولوژی‌های استفاده شده تکنولوژی‌های استفاده شده 27](#_Toc518837743)

[4-1- ابزار پرومتئوس 28](#_Toc518837745)

[۴-۱-۱- سیستم ایجاد هشدار در پرومتئوس 29](#_Toc518837746)

[۴-۲- ابزار انسیبل 31](#_Toc518837747)

[4-3- محیط‌تست ساوی 32](#_Toc518837748)

[4-4- زبان برنامه‌نویسی پایتون 32](#_Toc518837749)

[4-5- کتابخانه فلسک 33](#_Toc518837750)

[۴-۶- ردیس 34](#_Toc518837751)

[4-7- کتابخانه سِلِری 34](#_Toc518837752)

[۴-۸- کتابخانه پای‌مانگو 36](#_Toc518837753)

[۴-۹- دیتابیس مانگودی‌بی 36](#_Toc518837754)

[۵ فصل پنجم نحوه عملکرد سیستم‌توسعه‌پذیرکننده 38](#_Toc518837755)

[نحوه عملکرد سیستم‌توسعه‌پذیرکننده 39](#_Toc518837756)

[5-1- نحوه‌ی ساخت ماشین مجازی 39](#_Toc518837758)

[5-2- نحوه تعریف خوشه‌های کانتینری داکر سوارم 40](#_Toc518837759)

[5-3- مشاهده‌ی ماشین‌های مجازی موجود 41](#_Toc518837760)

[5-4- مشاهده‌ی خوشه‌های تعریف شده 41](#_Toc518837761)

[5-5- سناریو اول 42](#_Toc518837762)

[5-6- سناریو دوم 43](#_Toc518837763)

[۶ فصل ششم جمع‌بندي و نتيجه‌گيري و پیشنهاداتجمع‌بندي و نتيجه‌گيري 44](#_Toc518837764)

[منابع و مراجع 46](#_Toc518837765)

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

[شکل ‏1‑1- فلان بیسار 6](#_Toc518868106)

[شکل ‏1‑2 - مدل‌های سرویس در رایانش ابری 9](#_Toc518868107)

[شکل ‏1‑3- مقایسه‌های ماشین‌های مجازی با کانتینرها 12](#_Toc518868108)

[شکل ‏1‑4- نحوه‌ی کارکرد داکر 13](#_Toc518868109)

[شکل ‏0‑1- معماری داکر 20](#_Toc518868110)

[شکل ‏0‑2 - معماری کوبرنتیز 21](#_Toc518868111)

[شکل ‏0‑1 - معماری کلی نرم‌افزار 24](#_Toc518868112)

[شکل ‏0‑2- نمودار یوزکیس 25](#_Toc518868113)

[شکل ‏0‑3 - نمودار کانتکست 26](#_Toc518868114)

[شکل ‏0‑4 - نمودار آبشاری 28](#_Toc518868115)

[شکل ‏0‑1 - لوگوی ابزار پرومتئوس 32](#_Toc518868116)

[شکل ‏0‑2 - نحوه‌ی کارکرد مکانیزم هشدار در پرومتئوس 34](#_Toc518868117)

[شکل ‏0‑3 - لوگوی انسیبل 36](#_Toc518868118)

[شکل ‏0‑4 - لوگوی زبان برنامه‌نویسی پایتون 37](#_Toc518868119)

[شکل ‏0‑5 - لوگوی کتابخانه فلسک 37](#_Toc518868120)

[شکل ‏0‑6 - لوگوی ردیس 38](#_Toc518868121)

[شکل ‏0‑7 - قطعه کد تعریف عملیات ناهمگام در کلری 39](#_Toc518868122)

[شکل ‏0‑8- نحوه استفاده از کتابخانه پای‌مانگو 40](#_Toc518868123)

[شکل ‏0‑9 - نمایش یک سند نمونه در مانگو‌دی‌بی 41](#_Toc518868124)

[شکل ‏0‑1 - ایجاد سرور جدید 44](#_Toc518868125)

[شکل ‏0‑2 - تعریف خوشه‌ی کانتینری جدید 45](#_Toc518868126)

[شکل ‏0‑3 - مشاهده‌ی ماشین‌های مجازی موجود 45](#_Toc518868127)

[شکل ‏0‑4- مشاهده‌ی خوشه‌های کانتینری تعریف شده 45](#_Toc518868128)

[شکل ‏0‑5 - واکنش سیستم به ۷۰۰ تقاضا در ثانیه 46](#_Toc518868129)

[شکل ‏0‑6 – واکنش سیستم به سناریو شماره دوم 47](#_Toc518868130)

# ۱ فصل اول مقدمه

# مقدمه

خریداری یا استفاده از یک محصول با این پیش‌زمینه و تفکر که محصول مورد نظر نیاز خاصی را برطرف خواهد کرد، خود به خود انتظار برطرف کردن نیازمندی‌های ذهن مصرف‌کننده را در وی می‌انگیزد [1]. در ابتدا شاید صرفا رفع نیاز مصرف‌کنندگان، به هر روش ممکن، دغدغه اصلی تولیدکننده باشد اما به مرور و با گذشت زمان که نیازمندی‌ها پخته‌تر می‌شوند و ارتقا می‌یابند، کیفیت نیز در آن‌ها دخیل می‌شود. از طرفی، وجود نام‌ونشان‌های متعدد و متنوع در بسیاری از صنایع نیز، منجر به ایجاد رقابت میان فعالان هر عرصه شده است؛ رقابتی که کیفیت تعیین‌کننده‌ترین عامل برد و باخت در آن است [2]. صنعت نرم‌افزار نیز، به عنوان یکی از صنایع نوین که محصولاتش امروزه سهم قابل توجهی از بازار را در مصارف روزمره اداری و شخصی به خود اختصاص داده است، از این قاعده مستثنی نیست. بنابراین در تولید و توسعه یک محصول نرم‌افزاری نیز به منظور موفقیت هرچه بیشتر، می‌بایست به کیفیت، نگاه جدی داشته باشیم.

به طور خاص، در سامانه‌های کاربردی مبتنی بر وب (وب‌اپلیکیشن‌ها) و موبایل که جامعه کاربریشان هر روز بیشتر و بیشتر می‌شود، نیازمندی‌های مختلفی در طول چرخه عمر نرم‌افزار بروز پیدا می‌کنند. از طرفی در دنیای نرم‌افزار، گسترده‌تر شدن دامنه دسترسی به یک محصول نرم‌افزاری، الزاماتی برای آن فراهم می‌آورد که برای مثال، می‌توان گفت محصول نرم‌افزاری می‌بایست توسط یک فرد عادی از جامعه هدف مشتریان، قابل استفاده باشد. قابل استفاده بودن را نه در دانش فنی کاربران سیستم، بلکه در قابل فهم بودن رابط میان سیستم و کاربران تعریف می‌کنیم [۳].

البته ناگفته نماند دانش فنی و مهارت استفاده از ابزارهای فناوری‌محور، بخش غیرقابل اغماضی از توانایی استفاده از یک محصول نرم‌افزاری را ممکن می‌سازد؛ ولی امروزه، در مورد محصولات نرم‌افزاری تحت وب که به طور معمول با تعداد کاربران زیادی مواجه هستند، قابل استفاده بودن و کارایی[[1]](#footnote-1) آن‌ها در هنگام کار یک کاربر عادی، یکی از معیارهای مهم کیفیت به شمار می‌رود.

## تضمین و کنترل کیفیت

همانطور که پرسمن در کتابش [2] مطرح می‌کند، رسیدن به یک محصول با کیفیت در مهندسی نرم‌افزار، به صورت ضمنی و خود به خود ممکن نیست؛ بلکه نتیجه بازنگری در چهار بعد کلی در فرآیند مهندسی نرم‌افزار و اِعمال مجموعه آن‌ها است:

* روش‌های مهندسی نرم‌افزار
* تکنیک‌های مدیریت پروژه
* فعالیت‌های کنترل کیفیت
* فعالیت‌های تضمین کیفیت

طبق این اظهار نظر، با فرض اِعمال شدن روش‌های درست و بهره‌ور مهندسی نرم‌افزار و تکنیک‌های موثر در مدیریت پروژه تولید نرم‌افزار - که با تقریب خوبی هر دو را می‌توان جزو روش‌های مدیریتی و در حوزه تصمیم‌گیری‌های کلان سیستم دانست - بدیهی است که همچنان کنترل کیفیت و تضمین آن، دو بعد فنی و جزئی‌تر رسیدن به نرم‌افزار با کیفیت را تشکیل می‌دهند. بنابراین می‌بایست روش‌های موثر به منظور انجام فرایند‌های کنترل کیفیت و تضمین رسیدن به آن، توسط تیم مهندسی نرم‌افزار اتخاذ شود.

اما، مشابه هر فرایند و فعالیت دیگری، رسیدن به کیفیت نیز هزینه‌های خاص خود را دارد. هزینه کیفیت در نرم‌افزار، مطابق اظهارنظر پرسمن، به سه دسته هزینه‌های پیش‌گیری، هزینه‌های ارزیابی و هزینه‌های خرابی تقسیم می‌شود. هرکدام از این هزینه‌ها، در صورت پیش‌بینی و رفع نواقص محتمل/پیش‌آمده در هر مرحله از طراحی و پیاده‌سازی، بدون اینکه وارد مرحله بعدی شویم، می‌تواند با نرخ بسیار زیادی کاهش یابد [2].

یکی از علل عدم رضایت کاربران و مشتریان از وب‌اپلیکیشن‌ها - که درنتیجه این نارضایتی، آمار کاربران وب‌اپلیکیشن‌های کسب‌وکارها دستخوش تغییرات نامطلوب شده و حتی هزینه‌های گزافی به تیم مهندسی نرم‌افزار به خاطر اعمال تغییر پس از تحویل، وارد می‌شود- طراحی نه‌چندان کاربرپسندانه واسط کاربری و زیبایی آن‌هاست [4]؛ بدیهی است که استفاده از مدل‌های فرایندی چابک و تکراری می‌تواند در کاهش هزینه‌های طراحی مجدد پس از تحویل و یا اعمال تغییر در رابط‌های موجود، موثر باشد [2]، اما هنوز یک سوال بدون پاسخ خواهد ماند: «چه رابطی برای کاربران وب‌اپلیکیشن (محصول) من مناسب است و طبق نیازمندی‌های فعلی حداکثر کیفیت را تامین خواهد کرد؟» برای پاسخ به این سوال، چک‌لیست‌ها و توصیه‌های فراوانی [2], [5] ارائه شده است که هرکدام به نحوی در افزایش کیفیت رابط‌های کاربری تاثیرگذار بوده‌اند، اما برای تست یک رابط کاربری به صورت کمی، تحلیل و یافتن نقاط ضعف در زیبایی و همچنین ریزبینی در مورد کارایی یک واسط کاربری، به نظر می‌رسد که بررسی بیشتری مورد نیاز است [3].

## کارآیی

به تعبیر نویسندگان مرجع [3] هر کاربر می‌تواند برای خودش تعریفی از کارآیی ارائه نماید. در ادامه بررسی مفصل و مقایسه تطبیقی از مدل‌های کیفی مختلف و کارآیی در هرکدام انجام شده است ولی در اینجا به طور مختصر به ارائه و مقایسه سه نوع دیدگاه از تعریف کارآیی می‌پردازیم:

* سازمان بین‌المللی استانداردها (ایزو ۹۲۴۱-۱۱) کارآیی را در سه حوزه تعریف می‌کند:

«میزان سودی که استفاده از یک محصول در رسیدن به اهداف مورد نظر کاربران در رابطه با کاربردی مشخص، که همراه با تاثیرگذاری، بهره‌وری و رضایت باشد، کارآیی آن محصول نامیده می‌شود.»

* جامعه متخصصین کارآیی[[2]](#footnote-2) بیشتر روی فرایند تولید و توسعه محصول تمرکز می‌کنند و با بیان کارایی به عنوان «یک روش برای کاستن هزینه‌ها و تولید ابزارهایی که مختص کاربرانشان باشد»، از ویژگی مرتبط بودن همواره کارایی با کاربران، استفاده می‌کند.
* استیو کورگ در کتاب خود، کاری نکن که من به فکر کردن بیفتم تعریف عامیانه‌تری را ارائه می‌دهد؛ وی معتقد است که کارایی به معنی اطمینان حاصل کردن از کار کردن خوب محصول نهایی است. با این توضیح که یک فرد با دانش، توانمندی و تجربه کم نیز بایستی بتواند از محصول به راحتی استفاده کند و نیازهای خود را برطرف سازد.

با بررسی‌های مرجع [3] تمامی تعاریف مطرح برای کارایی، شامل سه زمینه کلیدی و مهم هستند:

1- کاربری وجود دارد.

2- این کاربر مشغول انجام کاری است.

3- کاربر، در حین انجام کار خود، با یک سیستم یا محصول نرم‌افزاری در تعامل است.

اینکه کاربر در طول دوره کاری‌اش با سیستم به طور دقیق به چه موارد منفی یا مثبت یا حتی خنثی برخورده، نقش مهمی در تجربه کاربری وی خواهد داشت؛ کارایی به طور کلی به توانایی کاربر در انجام موفق یک کار مشخص دلالت دارد، در حالی که تجربه کاربری به جنبه وسیع‌تری پرداخته و شامل احساسات، عواطف و ادراکات کاربر در حین کار با سیستم می‌شود [3].

در بخش‌های بعدی و با بررسی مدل‌های کیفی مختلف که به منظور سنجش کمی کیفیت نرم‌افزار ارائه شده‌اند، خواهیم دید که کارایی نرم‌افزار، به عنوان یکی از مشخصه‌های اصلی در اغلب این مدل‌ها و به صورت صریح بیان شده است. با بررسی پژوهش‌ها و کارهای گذشته و همچنین نکاتی از مرجع [2]، می‌توان گفت از سال ۱۹۷۰ تا به اکنون، تقریبا در هر مدل کیفی ارائه شده برای نرم‌افزار و به طور خاص برای سامانه‌های کاربری تحت وب، کارایی به صورت صریح به عنوان یک مشخصه اصلی بیان شده است؛ بنابراین می‌توان ادعا کرد کارایی یک نرم‌افزار، از جمله ویژگی‌های مهم کیفی در دستیابی و کنترل کیفیت نرم‌افزار است.

## کارایی و لایه‌های طراحی وب‌اپلیکیشن‌ها

کارایی در وب‌اپلیکیشن‌ها - که امروزه نقش مهمی در ارائه محتوا و سرویس به کاربران دارند - به عنوان یکی از ابعاد و مشخصه‌های اصلی و مهم در کیفیت مطرح است [2]. یکی از عوامل بسیار تاثیرگذار در کارایی هر محصولی، رابط کاربری آن است؛ همچنین بی‌کیفیتی و کارا نبودن یک محصولی که کاربر با آن در ارتباط است و کاربرد حساسی دارد، مانند تابلویی که در فلان قابل مشاهده است و در مرجع [3] از این حادثه یاد شده است، می‌تواند به مرگ و زندگی افراد ختم شود.



شکل ‏1‑1- فلان بیسار

با فراگیر شدن تکنولوژی‌های ابری نیاز به تغییر و مدیریت شرایط به صورت پویا کاربری بسیار زیادی پیدا کرده است. این به آن دلیل است که سرویس‌های ابری بایستی دائما بنابر شرایط و نیاز متقاضی منابع اختصاص یافته و بار سیستم را بین خوشه‌های کامپیوتری خود توزیع کنند. برای این کار نیاز به آن است که نرم‌افزارهایی که قرار است در محیط ابری اجرا شوند، از سخت‌افزار و سیستم‌عامل و معماری کامپیوتری که قرار است برروی آن اجرا شوند، منزوی[[3]](#footnote-3) باشند. به این منظور اخیرا مفهومی تحت عنوان کانتینرها سطح جدیدی از مجازی‌سازی را معرفی کرده است. در یک کانتینر تمام برنامه به همراه پیش‌نیاز‌های آن در محیطی منزوی قرار دارد که این کار جابجا کردن آن برنامه بین سیستم‌های مختلف را آسان کرده است.

نکته‌ی قابل توجه آن است که این کانتینرها به تنهایی نمی‌توانند نیازهای سیستم‌ها و هماهنگ‌کننده‌های ابری[[4]](#footnote-4) را برآورده کنند. به دلیل آن‌که مدیریت یک کانتینر پیچیدگی‌های خاصی دارد. از جمله این پیچیدگی‌ها می‌توان به بدست آوردن وضعیت فعلی اپلیکیشن (در حال اجرا،‌ متوقف شده و دارای خطا)، بروزرسانی بسته‌های نرم‌افزاری و زمان‌بندی[[5]](#footnote-5) آن‌ها برروی خوشه‌های کامپیوتری اشاره کرد. به منظور برطرف کردن این نیازها سیستم‌هایی تحت عنوان هماهنگ‌کننده کانتینر پدید آمده‌اند تا این نیازها را برطرف سازند. برای مثال می‌توان به داکر سوارم و کوبرنتیز به عنوان دو نمونه از معروف‌ترین هماهنگ‌کننده‌های کانتینری اشاره کرد.

یکی از مهم‌ترین فاکتورهای یک هماهنگ‌کننده‌ی کانتینری خوب توانایی به پاسخ‌گویی مناسب دربرابر تغییرات است. از جمله این تغییرات می‌توان به افزایش ناگهانی ترافیک و خارج شدن یک سرور از خوشه کامپیوتری اشاره کرد. هر دو هماهنگ‌کننده کانتینری اشاره شده در متن بالا این ویژگی‌ها را در خود دارند. اما پیاده‌سازی هر کدام از آن‌ها به نحوی است که برخی از نیاز‌های صنعتی را پاسخ می‌دهد. به عنوان مثال داکر سوارم عمل تقسیم بار شبکه را بین کانتینرها به صورت ایستا انجام می‌دهد. این بدین معناست که در صورت افزایش یا کاهش ترافیک خود فرد بایستی تعداد کانتینرها را تغییر دهد حال آن‌که در کوبرنتیز یک مجموعه‌ی رابط کاربری برنامه‌نویسی در اختیار توسعه قرار دهنده قرار می‌گیرد که از طریق آن می‌تواند نحوه واکنش سیستم در قبال رویداد‌های مختلف را تنظیم کرد. از جمله این رویدادها می‌توان به بار زیاد محاسباتی اشاره کرد. به عنوان مثال در صورتی که برنامه شما بار محاسباتی زیادی داشته باشد و قصد داشته باشید که این بار را بین خوشه‌های کامپیوتری توزیع کنید می‌توانید از این طریق یک حد آستانه برای آن تعریف کنید تا در صورتی که از آن حد آستانه فراتر رفت یک ماشین مجازی جدید به این خوشه شما اضافه کند. اما داکر سوارم فاقد رابط برنامه‌نویسی این چنینی می‌باشد و در صورت افزایش بار خود فرد بایستی که عمل توزیع پذیری را به صورت دستی انجام دهد.

هدف این پروژه کارشناسی پیاده‌سازی این رابط‌برنامه‌سازی برای داکر سوارم می‌باشد. به این صورت که شما می‌توانید همانند کوبرنتیز عمل‌های متنوعی را در قبال رویدادهای مختلف انجام دهید.

## مقدمه‌ای بر رایانش ابری

رایانش ابری شاخه‌ای از فناوری اطلاعات است که دسترسی همیشگی را به یک استخری از منابع قابل پیکربندی فراهم می‌آورد. هم‌چنین رایانش ابری امکان رصد کردن سرویس‌های ابری درحال اجرا برروی این منابع قابل پیکربندی با حداقل هزینه و زحمت ممکن می‌سازد. رایانش ابری مبتنی بر به اشتراک گذاری منابع به منظور دسترسی به یکپارچگی و فراهم‌کردن صرفه‌جویی در مقیاس مشابه پرداخت هزینه‌های برق برای منابع کامپیوتری می‌باشد.

### تاریخچه رایانش ابری

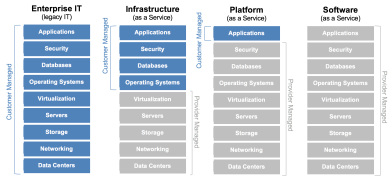
پیدایش مفاهیم اساسی رایانش ابری به دهه ۱۹۶۰ بازمی‌گردد. زمانی که جان مک‌کارتی اظهار داشت که «رایانش ممکن است روزی به عنوان یکی از صنایع همگانی سازماندهی شود». تقریباً تمام ویژگی‌های امروز رایانش ابری (تدارک الاستیک، ارائه به صورت یک صنعت همگانی، برخط بودن و توهم دسترسی به عرضه نامحدود) به همراه مقایسه با صنعت برق و شکل‌های مصرف عمومی و خصوصی و دولتی و انجمنی را پارک هیل داگلاس در کتابی که با عنوان «مشکل صنعت همگانی رایانه» در سال ۱۹۶۶ مورد بررسی قرار داد. واژه‌ی ابر در واقع بر گرفته از صنعت تلفن است به این‌گونه که کمپانی‌های ارتباطات راه دور که تا دهه ۱۹۹۰ تنها خطوط نقطه به نقطه‌ی اختصاصی ارائه می‌کردند، شروع به ارائه شبکه‌های خصوصی مجازی با کیفیتی مشابه و قیمتهای کمتر نمودند. نماد ابر برای نمایش نقطه مرزی بین بخش‌هایی که در حیطه مسئولیت کاربرند و آنهایی که در حیطه مسئولیت عرضه‌کننده بکار گرفته می‌شد. رایانش ابری مفهوم ابر را به گونه‌ای گسترش می‌دهد که سرورها را نیز علاوه برزیر ساخت‌های شبکه در برگیرد.

سایت آمازون با مدرن‌سازی مرکز داده خود نقش مهمی در گسترش رایانش ابری ایفا کرد. بعد از حباب دات-کام آن‌ها دریافتند که با تغییر مرکز داده‌های خود - که مانند اغلب شبکه‌های رایانه‌ای در بیشتر اوقات تنها از ۱۰٪ ظرفیت آن استفاده می‌شد و مابقی ظرفیت برای دوره‌های کوتاه اوج مصرف در نظر گرفته شده بود - به معماری ابر می‌توانند بازده داخلی خود را بهبود بخشند. آمازون از سال ۲۰۰۶ امکان دسترسی به سامانه خود از طریق وب سرویس‌های آمازون را بر پایه رایانش همگانی ارائه کرد. در سال ۲۰۰۷، گوگل و آی بی ام به همراه چند دانشگاه پروژه‌ای تحقیقاتی در مقیاسی بزرگ را در زمینه رایانش ابری آغاز نمودند.

در اواسط سال ۲۰۰۸ شرکت گارتنر متوجه وجود موقعیتی در رایانش ابری شد که برای «شکل دهی ارتباط بین مصرف‌کنندگان خدمات فناوری اطلاعات، بین آنهایی که این سرویس‌ها را مصرف می‌کنند و آن‌ها که این سرویس‌ها را می‌فروشند» بوجود می‌آید.

### مدل‌های سرویس در رایانش ابری

در این بخش به توضیح انواع مدل سرویس در رایانش ابری می‌پردازیم. مواد موجود برگرفته از ]۱[ می‌باشد



شکل ‏1‑2 - مدل‌های سرویس در رایانش ابری

#### زیرساخت به عنوان سرویس

زیرساخت به عنوان یک سرویس یک زیرساخت ضروری رایانش است که در محیط مجازی اینترنت ارائه و مدیریت شده است. مقیاس انعطاف پذیری دارد و به سرعت برای نیازهای زیاد و کم انطباق‌پذیر است همچنین پرداخت به اندازه آنچه استفاده می‌شود صورت می‌گیرد.

زیرساخت به عنوان سرویس به شما کمک می‌کند از هزینه و پیچیدگی خرید و مدیریت سرورهای فیزیکی خود و دیگر زیرساخت‌های مرکز داده (دیتا سنتر) جلوگیری کنید.

هر منبع به عنوان یک جزء خدمت جداگانه ارائه می‌شود و فقط لازم است شما تا زمانی که به آن نیاز دارید، یک منبع خاص را اجاره کنید. ارائه دهنده سرویس رایانش ابری در حین اینکه شما نرم افزارهای خود (سیستم عامل‌ها و برنامه‌های کاربری) خریداری، نصب، پیکربندی و مدیریت می‌کنید، زیرساخت را سامان‌دهی و مدیریت می‌کند.

#### پلتفرم به عنوان سرویس

پلتفرم به عنوان یک سرویس یک محیط رشد و توسعه کامل در رایانش ابری است با منابعی که شما را قادر می‌سازد همه چیز را از برنامه‌های کاربردی ساده ابری به برنامه‌های کاربردی سازمانی که پیچده تر هستند، تحویل دهید. شما منابع مورد نیاز از یک سرویس ارائه دهنده ابری را بر اساس پرداختی که انجام می‌دهید خریداری می‌کنید و از طریق یک اتصال ایمن اینترنت به آن‌ها دسترسی دارید.

مانند زیرساخت به عنوان سرویس، پلتفرم به عنوان سرویس شامل زیرساخت‌هایی مانند: سرورها، ذخایر و شبکه می‌شود که علاوه بر آن شامل نرم‌افزارهای ارتباطی، ابزارهای توسعه، خدمات هوش تجاری، سیستم‌های مدیریت پایگاه داده و... نیز می‌شود. پلتفرم به عنوان سرویس برای پشتیبانی از یک چرخه حیات کامل برنامه‌های کاربردی وب طراحی شده است این چرخه عبارت است از: ساخت، آزمایش، رشد و توسعه، مدیریت و به‌روزرسانی.

پلتفرم به عنوان سرویس به شما این امکان را می‌دهد که هزینه و پیچیدگی خرید و مدیریت لایسنس نرم‌افزارها، زیرساخت برنامه‌های کاربردی بنیادی، نرم‌افزار‌های ارتباطی، ابزارهای توسعه و سایر منابع را از بین ببرید. شما برنامه‌های کاربردی و خدماتی را که توسعه می‌دهید، مدیریت می‌کنید و ارائه دهنده سرویس ابری به طور معمول سایر چیزها را مدیریت می کند.

#### نرم‌افزار به عنوان یک سرویس

نرم‌افزار به عنوان یک سرویس به کاربران اجازه می‌دهد به برنامه‌های کاربردی مبتنی بر ابر در فضای مجازی اینترنت متصل شده و از آن‌ها استفاده کنند. مثال‌های رایج در این زمینه عبارتند از: ایمیل، تقویم و ابزارهای آفیس.

نرم‌افزار به عنوان سرویس یک راه‌حل نرم‌افزاری کامل ارائه می‌دهد که شما براساس پرداختی که به ارائه‌دهنده سرویس ابری انجام می‌دهید، آن را خریداری می‌کنید.

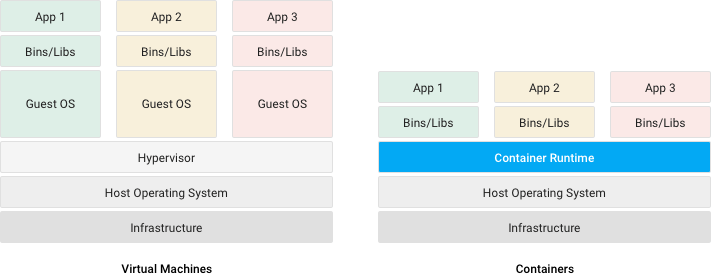
شما استفاده از برنامه کاربردی را برای سازمان و کاربران خود که معمولاً به‌وسیله یک مرورگر در فضای اینترنت به آن متصل می‌شوند، اجاره می‌کنید. همه زیرساخت‌های بنیادی، نرم‌افزارهای ارتباطی، نرم‌افزارها و داده‌های برنامه‌ کاربردی، در مرکز داده (دیتاسنتر) ارائه دهنده سرویس قرار داده شده است. ارائه‌ دهنده سرویس، نرم‌افزار و سخت‌افزار را مدیریت می‌کند و با شرایط خدمات مناسب قابلیت دسترسی و امنیت برنامه و داده‌های شما تضمین خواهد کرد.

نرم‌افزار به عنوان سرویس به سازمان شما این امکان را می‌دهد با یک برنامه کاربردی مقرون به صرفه به سرعت پیشرفت کند و امور خود را اداره کند.

## مقدمه‌ای بر کانتینرها

کانتینرها مکانیزم منطقی بسته‌بندی ارائه می‌دهند که در آن برنامه‌های کاربردی می‌توانند از محیطی که در آن در حال اجرا هستند، جداسازی شوند. این جداسازی باعث می‌شود تا برنامه‌های کاربردی مبتنی بر کانتینرها به راحتی دیپلوی شوند بدون در نظر گرفتن آن‌که محیط هدف یک مرکز داده خصوصی است یا کامپیوتر شخصی توسعه‌دهنده. کانتینرایز کردن جدسازی دغدغه‌ها را به زیباترین وجه ممکن انجام می‌دهد به این صورت که توسعه‌دهندگان تنها بر منطق برنامه و وابسته‌های آن فکر می‌کنند و تیم‌های عملیات تکنولوژي اطلاعات برروی قراردادن آن و مدیریت آن تمرکز دارند.

برای کسانی که اغلب از محیط‌های مجازی‌سازی شده می‌آیند، کانتینرها معمولا با ماشین‌های مجازی مقایسه می‌شوند. شما ممکن است با ماشین‌های مجازی آشنا باشید: یک سیستم عامل میهمان مانند لینوکس یا ویندوز برروی یک سیستم‌عامل میزبان با دسترسی‌های مجازی‌سازی شده به سخت‌افزار. همانند ماشین‌های مجازی، کانتینرها به شما این امکان را می‌دهند که برنامه خود را با تمام کتابخانه‌ها و وابسته‌هایش پکیج بندی کنید. این کار یک محیط منزوی برای اجرا سرویس‌های نرم‌افزاری را در اختیار شما قرار می‌دهد. همانطور که شما می‌توانید پایین‌تر مشاهده کنید، شباهت‌ها در اینجا پایان می‌پذیرد چون کانتینرها واحدهای بسیار سبک‌تری برای توسعه‌دهندگان و تیم‌های عملیات فناوری اطلاعات فراهم می‌آورند که مزایای بسیاری را به همراه دارد.



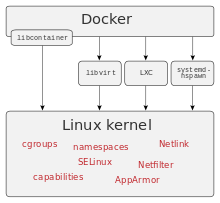
شکل ‏1‑3- مقایسه‌های ماشین‌های مجازی با کانتینرها

برای اجرای کانتینرها به صورت آسان و ساده معمولا احتیاج به یک موتور کانتینر می‌باشد در بخش بعدی به معرفی موتور کانتینر داکر می‌پردازیم.

## داکر چیست؟

داکر ابزاری است که ایجاد، اجرا و دیپلوی کردن برنامه‌های کانتینری را آسان کرده است. داکر یک رابط برنامه‌نویسی سطح بالا فراهم کرده است که با استفاده از آن می‌توان کانتینرهای سبک را در محیط‌های مجزا اجرا کرد.

با ساخته شدن برروی امکاناتی که توسط هسته‌ی لینوکس فراهم شده است (به طول عمده سی گروپ‌ها) یک کانتینر داکر، برخلاف یک ماشین مجازی، دارای یک سیستم‌عامل مجزا نمی‌باشد. برخلاف آن، این برروی کاربردهای هسته‌های سیستم‌عامل تمرکز می‌کند و از منزوی‌سازی منابع برای سی‌پی‌یو و حافظه استفاده می‌کند. هم‌چنین از فضای‌نام‌های جداگانه برای جداکردن دید اپلیکیشینی سیستم‌عامل استفاده می‌کند.



شکل ‏1‑4- نحوه‌ی کارکرد داکر

## هماهنگ‌کننده‌ی کانتینری[[6]](#footnote-6) چیست؟

هماهنگ‌کننده کانتینری به کاربران اجازه می‌دهد تا کانتنیرها را با یکدیگر در فضای ابری هماهنگ کنند. به عنوان مثال، کاربران می‌توانند یک خوشه‌ی ردیس[[7]](#footnote-7) درست کنند که توسعه‌پذیر و قابل اطمینان است و شامل نودهای مستر ردیس و نودهای اسلیو ردیس می‌باشد. هماهنگ‌کننده کانتینری نه تنها نحوه قرارگیری اولیه کانتینرها را مشخص می‌کند، بلکه مدیریت چندین کانتینر را به عنوان یک مفهوم واحد نیز پشتیبانی می‌کند. مدیریت چندین کانتینر شامل دسترس پذیری، توسعه‌پذیری و شبکه‌ی کانتینرها می‌باشد. به عنوان مثال به جای مدیریت تک تک کانتینرها در یک خوشه‌ی ردیس، شما می‌توانید تمام کانتینرهای داخل یک خوشه را باهم مدیریت کنید. به عنوان مثال به دو نمونه معروف از هماهنگ‌کننده‌های کانتینری می‌توان به داکر سوارم[[8]](#footnote-8)، کوبرنتیز[[9]](#footnote-9) اشاره کرد.

## تعریف مساله

همانطور که در بخش قبل توضیح داده شد، یکی از ویژگی‌هایی که یک هماهنگ‌کننده کانتینری فراهم می‌کند، توسعه پذیری می‌باشد. پیاده‌سازی این توسعه‌پذیری در هماهنگ‌کننده‌های کانتینری به روش‌های مختلفی می‌باشد. برخی از هماهنگ‌کننده‌های کانتینری مانند داکر سوارم این توسعه‌پذیری را به صورت ایستا فراهم می‌کنند. این بدین معناست که مسئول بخش عملیات‌های فناوری اطلاعات بایستی یک خوشه‌ی کانتینری که متناسب با بار برروی سیستم می‌باشد، ایجاد کند و سپس خود داکر سوارم عمل تقسیم را بین این گره‌ها انجام می‌دهد. حال آنکه در صورت تغییر وضعیت و افزایش یا کاهش بار، داکر سوارم هیچ عملی را در راستای تطبیق با بار جدید انجام نمی‌دهد و نیازمند آن است که خود مسئول بخش عملیات‌های فناوری اطلاعات به صورت دستی اقدام به بازچینش خوشه‌ی کانتینری نماید و آن را مطابق با نیازمندی‌های جدید تغییر دهد.

در این پروژه کارشناسی قصد داریم تا این عملیات را به صورت اتوماتیک برمبنای پارامتری که مسئول عملیات تعیین می‌نماید انجام دهیم. لازم به ذکر است که این‌کار را برای هماهنگ‌کننده‌ی کانتینری داکر سوارم انجام می‌دهیم.

# ۲ فصل دوم مفاهیم اولیه



# ۲- مفاهیم اولیه

در این بخش به توضیح مفاهیم مربوط به توسعه‌پذیری می‌پردازیم و سپس دو هماهنگ‌کننده‌ی کانتینری را توضیح داده و معماری آن‌ها را بررسی می‌نماییم.

## رویکردهای توسعه‌پذیری

در این بخش در ابتدا به رویکردهای توسعه‌پذیری می‌پردازیم و سپس علت استفاده از رویکرد توسعه‌پذیری خودکار را مطرح می‌نماییم.

رویکردهای متداول نسبت به توسعه‌پذیری به سه روش عمده تقسیم می‌شوند:

* توسعه‌پذیری خودکار
* توسعه‌پذیری زمان‌بندی شده
* توسعه‌پذیری پیش‌گویانه

### توسعه‌پذیری خودکار

توسعه‌پذیری خودکار در رایانش ابری به این صورت تعریف می‌شود که تعداد کامپیوترهای فعال به صورت اتوماتیک برمبنای باری که روی مزرعه‌ی سروری قرار دارد، تنظیم می‌شود. این مفهوم بسیار نزدیک به ایده‌ی تقسیم بار[[10]](#footnote-10) می‌باشد.

توسعه پذیری خودکار مزایای زیر را فراهم می‌کند:

* برای شرکت‌هایی که زیرساخت وب‌سرور را خودشان فراهم می‌کنند، این بدین معناست که آن‌ها می‌توانند برخی از سرورها را در زمان کاهش ترافیک از دور خارج کنند و در هزینه‌هایی مانند برق صرفه‌جویی نمایند.
* برای شرکت‌هایی که از زیرساخت‌های ابری استفاده می‌کنند، توسعه‌پذیری خودکار به معنای صورت‌حساب‌های کمتر است. چون اغلب شرکت‌هایی که خدمات ابری ارائه می‌دهند، هزینه‌ی دریافتیشان برمبنای مجموع مصرف می‌باشد و نه بیشینه‌ی مصرف.
* راه‌حل‌های توسعه‌ی خودکار نظیر آن‌که توسط شرکت آمازون ارائه می‌شود، می‌تواند نمونه‌های ناسالم را جایگزین کند و دربرابر شکست‌های شبکه، سخت‌افزار و برنامه‌های کاربردی محافظت نماید.
* توسعه‌پذیری خودکار زمان دسترس‌پذیری محصول را افزایش می‌دهد برای زمانی که بار قرارگرفته روی محصول مشخص نمی‌باشد.

توسعه‌پذیری خودکار یک رویکرد واکنشی در مواجه با ترافیک می‌باشد. گاهی اوقات که تغییرات بسیار سریع رخ می‌دهند، این رویکرد ممکن است ناکارآمد باشد. در ادامه دو رویکرد دیگری که برای توسعه‌پذیری استفاده می‌شود را بیان می‌کنیم.

### توسعه‌پذیری زمان‌بندی شده

در توسعه‌پذیری زمان‌بندی شده برخلاف توسعه‌پذیری خودکار، توسعه در زمان‌های مشخصی روی خواهد داد. به عنوان مثال فرض کنید که در ساعت مشخصی از روز قرار است که بارسیستم به صورت ناگهانی افزایش پیدا کند. برای پاسخگویی به این حجم از بار ساعت مشخص را تعیین کرده و دستور به افزایش تعداد گره‌ها به منظور پاسخگویی به این بار می‌دهید. این رویکرد برای افزایش بارهای قابل پیش‌بینی مناسب است.

### توسعه‌پذیری پیش‌بینی کننده

در این روش ایده آن است که میزان بار سیستم را در زمان‌های گذشته ذخیره‌سازی کرده و از آن برای پیش‌بینی بار در آینده استفاده کند. به عنوان مثال شرکت نتفلیکس، در بخشی از زیرساخت‌های خود به این نتیجه رسید که موتور پیشگوینده‌ی آن اسکریر[[11]](#footnote-11)، نتایج بهتری از توسعه‌پذیر خودکار آمازون ارائه می‌داد. به صورت مشخص در بخش‌های زیر بهتر عمل می‌کرد:

* تشخیص ضربه‌های ناگهانی و بزرگ در آینده و آماده‌سازی ظرفیت برای این ضربه‌ها از پیش
* مدیریت خروج‌های ناگهانی و در ابعاد بزرگ، مانند از دست رفتن یک ناحیه به صورت کلی
* مدیریت ترافیک‌های متغیر و ارائه‌ی انعطاف در مقابل این ترافیک متغیر

## هماهنگ‌کننده‌های کانتینری و ویژگی‌های آن

در بخش مقدمه توضیحات کلی درباره‌ی هماهنگ‌کننده‌های ابری داده شد. در اینجا به صورت دقیق‌تر به توضیح درباره‌ی دو هماهنگ‌کننده‌ی کانتینری معروف و معماری آن‌ها می‌پردازیم.

### هماهنگ‌کننده کانتینری داکر سوارم

داکر سوارم یک هماهنگ‌کننده‌ی ابری می‌باشد که در داخل داکر قرار داده شده است. ویژگی‌های زیر را فراهم می‌کند:

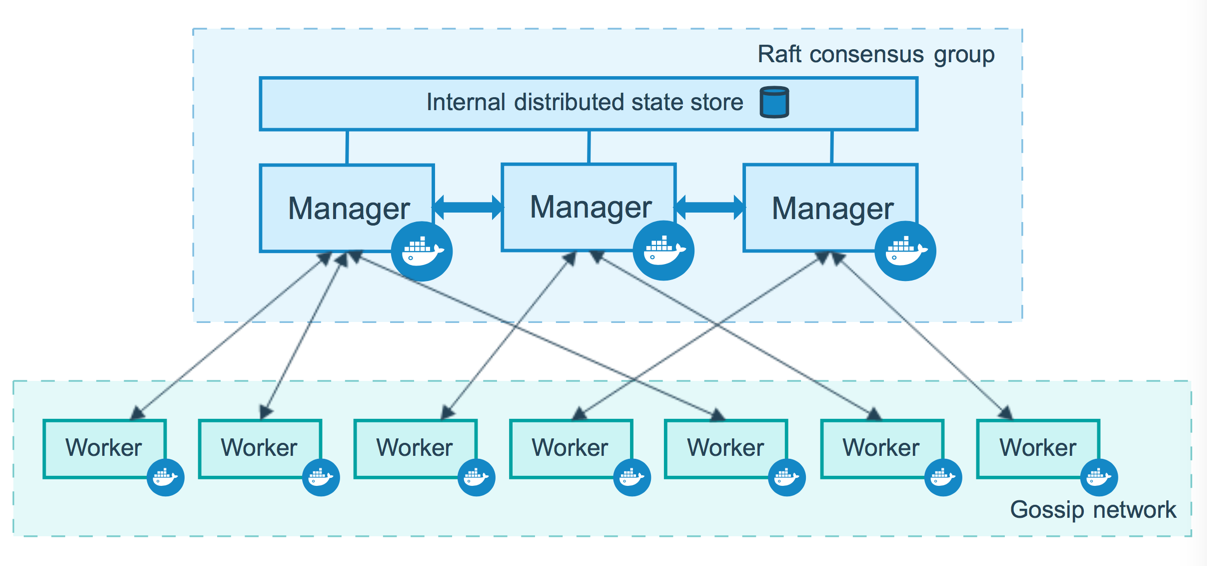
* قابلیت مدیریت خوشه‌های کانتینری تنها با نصب داکر
* طراحی توزیع شده
* مدل سرویس اعلامی[[12]](#footnote-12)
* توزیع‌پذیری
* کشف سرویس
* تقسیم بار

یکی از مهم‌ترین ویژگي‌هایی که داکر سوارم فراهم مي‌کند امکان توزیع‌پذیری می‌باشد. این بدین معناست که شما می‌توانید تعداد تکرار سرویس خود را در یک خوشه‌ی کامپیوتری تنظیم نمایید. از دیگر مسائل مهم در خوشه‌ی کانتینری داکر سوارم، نوع گره‌ها می‌باشد. در این خوشه گره‌ها دو نوع دارند:

* گره‌های مدیر: گره‌های مدیر وظیفه‌ی نگه‌داری وضعیت خوشه را برعهده دارند. هم‌چنین دیگر وظیفه‌ی این گره‌ها زمان‌بندی سرویس‌ها روی بقیه گره‌ها می‌باشد. علاوه بر این وظایف گره‌های مدیر رابط برنامه‌نویسی سوارم را فراهم می‌نمایند.
* گره‌های کارگر: تنها وظیفه‌ی گره‌های کارگر اجرای کانتینر ها می‌باشد.

یکی از مسائل مهم در سیستم‌های توزیع‌شده انتخاب گره رهبر می‌باشد. این کار معمولا از طریق الگوریتم‌های سرشماری صورت می‌پذیرد. در داکر سوارم از پیاده‌سازی رفت[[13]](#footnote-13) برای این کار استفاده می‌شود. از ویژگی های مهم دیگر استفاده از این الگوریتم آن است که در صورتی که یکی از گره‌هایی که وظیفه‌ی مدیریت خوشه را برعهده دارد از کار خارج شود، گره دیگری می‌تواند این مسئولیت را برعهده بگیرد و جلوی از دسترس خارج شدن سیستم را بگیرد. برای استفاده از این ويژگی، داکر سوارم پیشنهاد می‌کند که از تعداد فردی گره مدیریت‌کننده استفاده نمایید.

در شکل زیر می‌توانید معماری کلی داکر سوارم را مشاهده نمایید ]۵[.



شکل ‏0‑1- معماری داکر

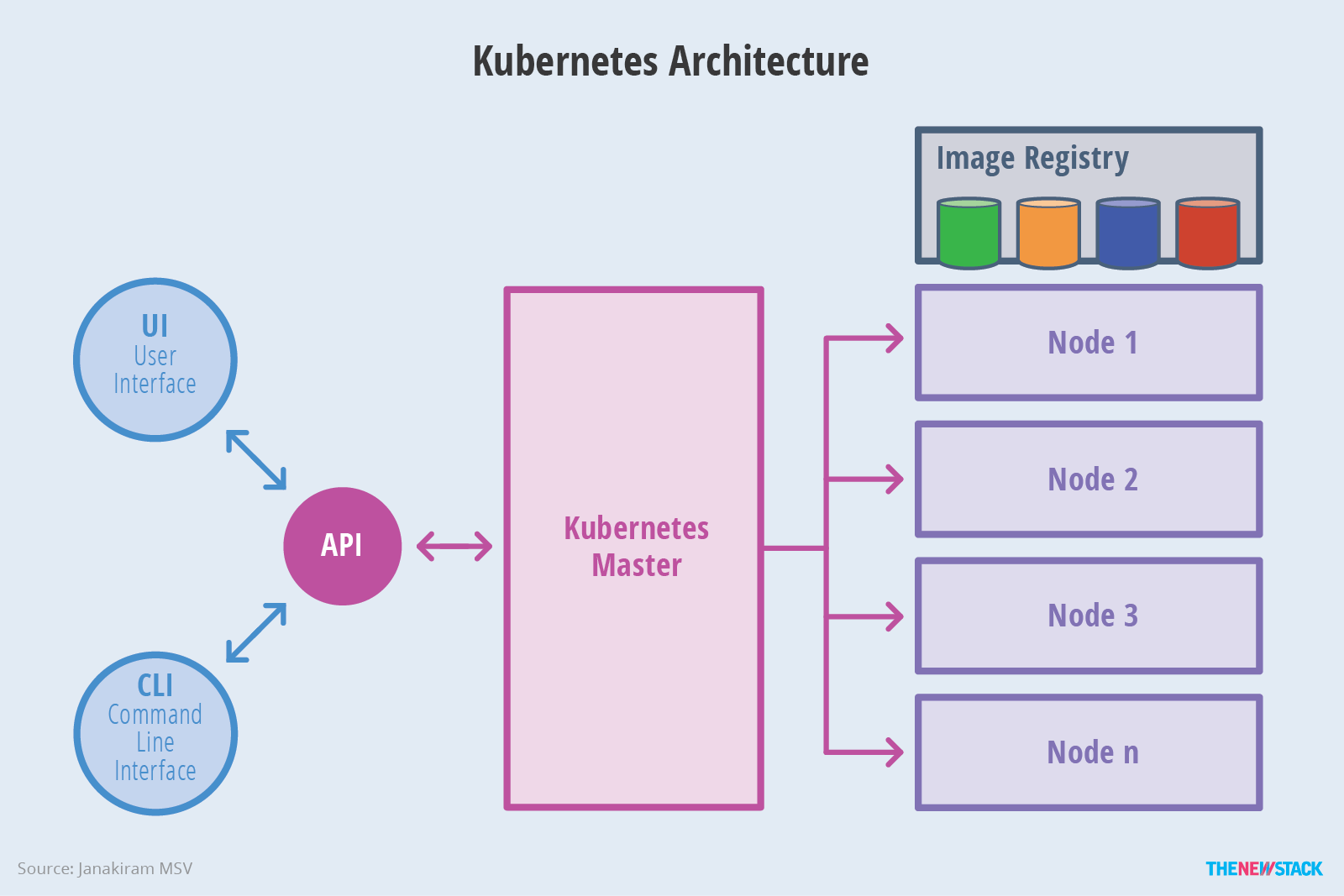
### هماهنگ‌کننده‌ی کانتینری کوبرنتیز

کوبرنتیز یک هماهنگ‌کننده‌ی کانتینری متن باز می‌باشد که برای اجرا و مدیریت کانتینر‌ها در سطح صنعت توسعه داده شده است. کوبرنتیز برمبنای ۱۵ سال تجربه‌ی گوگل در ساختن کانتینرها بنا شده است. ]۲[

از ویژگی‌های کوبرنتیز می‌توان به موارد زیر اشاره کرد:

* بسته‌بندی خودکار: با توجه به نیازمندی و منابع موردنیاز هر کانتینر، کوبرنتیز به صورت اتوماتیک کانتینرهای شما را روی گره‌ها قرار می‌دهد.
* ترمیم خودکار: کوبرنتیز به صورت خودکار تمامی کانتینرهایی را که موفق به اجرا نشوند را بازنشانی می‌کند. هم‌چنین در صورتی که گره‌ای که کانتینر برروی آن در حال اجرا است، دچار مشکل شود، کوبرنتیز تمامی کانتینرهای در حال اجرا برروی آن را مجددا روی گره‌های باقی‌مانده قرار می‌دهد.
* توسعه‌پذیری افقی: کوبرنتیز این امکان را به شما می‌دهد تا برنامه‌کاربردی خود را به صورت اتوماتیک کوچک یا بزرگ کنید. هم‌چنین این امکان را به شما می‌دهد. تا این کار را برعهده‌ی خود کوبرنتیز قرار دهید تا این کار را به صورت خودکار انجام دهد.
* کشف سرویس و توزیع‌بار:‌ کوبرنتیز بدون احتیاج به تغییر اپلیکیشن شما به شما این امکان را می‌دهد تا سرویس خود را ثبت‌نام کنید و اجازه دهید تا بقیه سرویس‌ها از وجود آن خبر دارد شوند. کوبرنتیز به هر کانتینر یک آدرس آی‌پی اختصاص می‌دهد و برای هر گروه از کانتینرها یک نام دی‌ان‌اس اختصاص می‌دهد که امکان توزیع بار بین آن‌ها را می‌دهد.

معماری کلی کوبرنتیز را می‌توانید در شکل زیر مشاهده کنید.



شکل ‏0‑2 - معماری کوبرنتیز

همانطور که در این شکل مشاهده می‌کنید، کوبرنتیز نیز مانند داکر سوارم از یک مجموعه‌ی گره تشکیل شده است که این گره‌ها می‌توانند نقش مستر و یا کارگر را برعهده داشته باشند. تنها گره‌های مستر توانایی اجرای عملیات زمان‌بندی کانتینرها برروی گره‌های کارگر یا خودشان را دارا می‌باشند. هر گره مستر یک رابط برنامه‌نویسی دارد که از طریق آن با رابط کنسولی و رابط تحت‌وب می‌توان ارتباط برقرار نمود. از مهم‌ترین تفاوت‌های کوبرنتیز با داکر سوارم می‌توان به توانایی توسعه‌پذیری در مقایس افقی به صورت اتوماتیک اشاره کرد. ما در این پروژه قصد داریم مشابه با این توسعه‌پذیری خودکار برای داکر سوارم پدید آوریم.

# ۳ فصل سوم طراحی و پیاده‌سازی

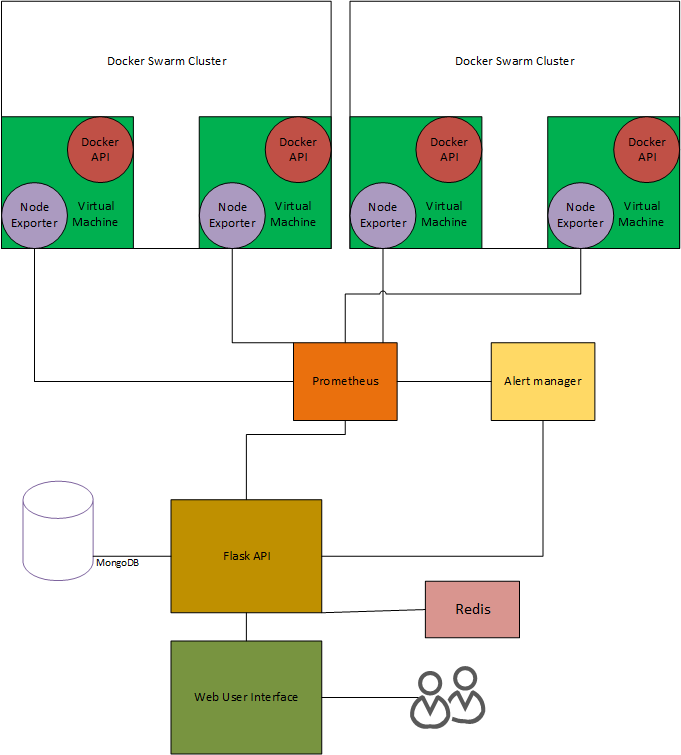


# طراحی و پیاده‌سازی

در این فصل به توضیح نحوه‌ی پیاده‌سازی و معماری کلی سیستم می‌پردازیم. سپس متدولوژی استفاده شده برای توسعه این نرم‌افزار را مورد بحث قرار خواهیم داد.

## شرح کلی طراحی

در حال حاضر پروژه تنها یک کاربر دارد و کاربر موردنظر به تمامی بخش‌های سیستم دسترسی دارد. معماری کلی سیستم به صورت شکل زیر می‌باشد:



شکل ‏0‑1 - معماری کلی نرم‌افزار

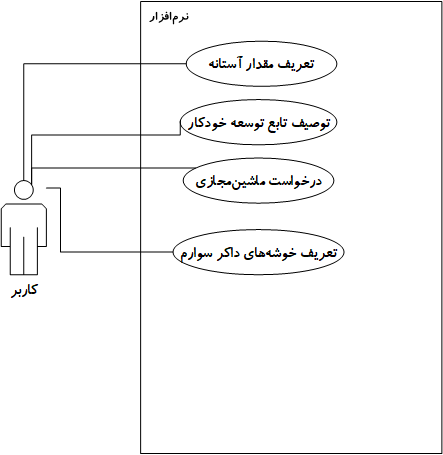
در بخش‌های بعدی توضیحات تکمیلی‌تر در رابطه با معماری نرم‌افزار داده خواهد شد.

## نمودارهای تحلیل

در این بخش به توضیح نمودارهای تحلیل و طراحی می‌پردازیم. از کتاب پرسمن به عنوان مرجع مهندسی نرم‌افزار در این پروژه استفاده شده است. ]۴[

### نمودار یوزکیس

در اینجا از نمودار یوزکیس[[14]](#footnote-14) برای نشان دادن نیازمندی‌های پروژه استفاده می‌شود. در اینجا تنها یک نقش داریم و آن نقش کاربری است که از سیستم استفاده می‌کند.



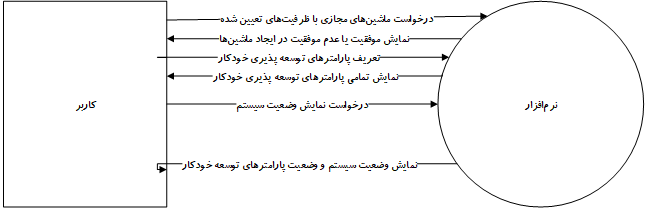
شکل ‏0‑2- نمودار یوزکیس

نیازمندی‌های این کاربر عبارت از موارد زیر می‌باشد:

* درخواست ماشین مجازی: کاربر می‌تواند از این سیستم درخواست ماشین مجازی نماید. سیستم در حال حاضر از اوپن استک[[15]](#footnote-15) به عنوان یک فراهم‌کننده‌ی ابری پشتیبانی می‌کند. اما ساختار کد به نحوی می‌باشد که اضافه کردن فراهم‌کننده‌های ابری دیگر نیز آسان می‌باشد. هنگامی که شما در خواست ماشین مجازی از این سیستم می‌نمایید، سیستم به شما این امکان را می‌دهد که با آن خوشه‌ی کانتینری بسازید. هم‌چنین برروی این ماشین مجازی نرم‌افزارهای موردنیاز مانند داکر و پرومتئوس نصب می‌گردد.
* تعریف خوشه‌های کانتینری: پس از تعریف ماشین‌های مجازی مربوطه، سیستم به شما این امکان را می‌دهد که خوشه‌های کانتینری براساس ماشین‌های مجازی‌ای که ساخته‌اید تعریف کنید.
* تعریف مقدار آستانه: در این بخش سیستم به شما اجازه می‌دهد تا پارامتر موردنظر برای توسعه‌پذیری، که تقریبا می‌تواند هر پارامتری باشد، را تعریف کنید. پس از تعریف این پارامتر سیستم با رصد دائمی وضعیت در صورت رخداد آن حادثه یک ماشین مجازی به سیستم اضافه می‌کند.

### نمودار کانتکست

نمودار کانتکست آن به صورت زیر می‌باشد:



شکل ‏0‑3 - نمودار کانتکست

با توجه به توضیحات داده شده درباره‌ی نرم‌افزار و نمودار یوزکیس به نظر نمودار کانتکست احتیاجی به توضیح نداشته باشد.

## متدلوژوی مورد استفاده

متدلوژی عبارت است مدل‌هایی که برای توصیف رهیافت‌های توسعه نرم‌افزار استفاده می‌شود. در مهندسی نرم‌افزار، الگوهای متعددی تعریف شده است. یکی از این الگوها، آبشاری نام دارد و در این پروژه از این الگو استفاده شده است. در ادامه درباره جزییات و ویژگی‌های الگوی آبشاری توضیح داده خواهد شد.

### الگوی آبشاری

الگوی آبشاری فرایندها را به گونه‌ای نشان می‌دهد که کجا تولیدکنندگان نرم‌افزار (برنامه‌نویسان) فازهای زیر را به ترتیب انجام دهند:

* مشخصات مورد نیاز (تحلیل نیازمندی‌ها)
* طراحی نرم‌افزار
* پیاده‌‌سازی و یکپارچه‌‌سازی
* تست نرم‌افزار (یا اعتبارسنجی)
* گسترش نرم‌افزار (یا نصب)
* نگه‌داری نرم‌افزار



شکل ‏0‑4 - نمودار آبشاری

در این مدل، فعالیت‌های تولید نرم‌افزار در قالب فازهایی با توالی مشخص و به ترتیب، برنامه‌ریزی و اجرا می‌شوند. اشکال عمده این روش این است که بازبینی و تجدید نظر در فازهای انجام شده امکان‌پذیر نیست. لذا خطای تخمین ابعاد پروژه، ریسک اشتباه در فهم درست و تحلیل نیازمندی‌ها و نیز امکان انتخاب نابجای معماری بسیار بالا می‌باشد.

در سختگیرانه‌ترین حالت آبشاری، بعد از اینکه هر فاز کاملاً پایان پذیرفت، به مرحله بعدی می‌رویم. بازبینی که اجازه ایجاد تغییرات در سامانه را بدهد (که ممکن است شامل تغییرات فرایندهای کنترل رسمی باشد) فقط قبل از رفتن به مرحله بعد امکان‌پذیر است. همچنین بازبینی ممکن است جهت اطمینان از پایان قطعی این فاز (مرحله) بکار گرفته شود. فازی که معیارهای تکمیل آن کامل شده باشد، معمولاً با عنوان دروازه اطلاق می‌شود که نشان می‌دهد پروژه از فاز فعلی به فاز بعدی منتقل شده‌ است. الگوی آبشاری از بازبینی و تجدید نظر فازهای قبلی که کامل شده‌اند، جلوگیری می‌کند. این عدم انعطاف‌پذیری مفصل در الگوی آبشاری محض، دست‌مایه انتقاد پشتیبانی‌کنندگان الگوهای انعطاف‌پذیر است.

مدل آبشاری یک مدل ترتیبی توسعه و تولید نرم‌افزار است و درآن مراحل تولید به شکل یک جریان مداوم متمایل به سمت پایین است (همانند یک آبشار) که شامل فازهای تحلیل خواسته‌ها، طراحی، پیاده‌سازی، آزمودن و تست کردن، یکپارچه سازی، و دادن محصول به بازار می‌شود. اغلب گفته می‌شود ریشه‌ی اصطلاح آبشاری از مقاله‌ای گرفته شده ‌است که توسط وینستون واکر رویس در سال ۱۹۷۰ نوشته شده ‌است. مدیریت و مراحل تکمیل پروژه در این متدولوژی به سادگی قابل پیاده‌سازی است. زیرا در مرحله اول که مرحله بررسی نیازمندی‌های پروژه می‌باشد، مشتری و تیم برنامه‌نویسی طی چند جلسه به بررسی نیازمندی‌ها و خواسته‌های پروژه می‌پردازند. پس از آن نوبت به مرحله طراحی می‌رسد. در مرحله‌ طراحی، افراد طرح کلی پروژه را می‌ریزند و جزییات پیاده‌سازی مشخص می‌شود. پس از مرحله‌ طراحی،   
تیم برنامه‌نویسی، خود را برای پیاده‌سازی آماده می‌کند. در این مرحله همه قسمت‌های کد، پیاده‌سازی می‌شوند. در انتهای این مرحله، ما مرحله یکپارچه‌سازی را خواهیم داشت که یکی از مشکل‌ترین قسمت‌های انجام پروژه‌ها در این مرحله می‌باشد. زیرا تنوع و گستردگی کار کامل در این مرحله نقش دارد. هر چه میزان گستردگی کار بالاتر باشد، سختی یکپارچه‌سازی نیز بیشتر خواهد بود.

#### تحلیل و تعریف خواسته ها

خدمات سیستم، محدودیت‌ها و اهداف از طریق مشورت با کاربر یا کاربران مشخص می‌شوند. این‌ موارد به طور مشروح تعریف می‌شوند و به صورت مشخصات سیستم مورد استفاده قرار می‌گیرند.

#### طراحی سیستم و نرم افزار

فرآیند طراحی سیستم‌ها، خواسته‌ها را به سیستم‌های نرم‌افزاری و سخت‌افزاری تقسیم می‌کند.   
بدین ترتیب، یک معماری کلی به وجود می‌آید. طراحی نرم‌افزار شامل شناسایی و توصیف انتزاع‌های سیستم نرم‌افزار و روابط آن‌هاست.

#### پیاده سازی و تست واحد

در این مرحله، طراحی نرم‌افزار به صورت مجموعه‌ای از برنامه‌ها و یونیت‌های جدا از هم در می‌آیند. در تست واحد، بازبینی می‌شود که آیا هر واحد خواسته‌های مورد نظر را برآورده می‌کند.

#### جامعیت و تست سیستم

واحدهای اولیه برنامه یا برنامه‌ها جامعیت پیدا می‌کنند و به عنوان یک سیستم کامل تست می‌شود تا تضمین شود که خواسته‌های نرم افزار برآورده شده‌اند. پس از تست، سیستم نرم‌افزار به مشتری تحویل داده می‌شود.

#### به کارگیری و نگهداری

این مرحله، معمولاً طولانی‌ترین مرحله چرخه حیات نرم‌افزار است. سیستم نرم‌افزاری نصب و به کار گرفته می‌شود. نگهداری، شامل تصحیح خطاهایی است که در مراحل اولیه چرخه حیات برطرف نشدند. همچنین شامل بهبود پیاده‌سازی‌ واحدهای سیستم و اصلاح خدمات سیستم جهت پاسخگویی به نیازهای جدید نیز می‌باشد.

## ابزار مورد استفاده

زبان مورد استفاده برای توسعه پایتون می‌باشد. استفاده از این زبان به علت گستردگی کتابخانه‌های موجود برای این مقصود خاص پروژه می‌باشد. هم‌چنین داکر نیز تنها کتابخانه رسمی‌اش به زبان پایتون می‌باشد. از ویرایش‌گر فایل‌های متنی ویژوال استودیو کد[[16]](#footnote-16) برای توسعه این نرم‌افزار استفاده شده است. از سیستم مدیریت وابسته‌های پیپ‌اِنو[[17]](#footnote-17) استفاده شده است. از ویژگی‌های این سیستم مدیریت وابسته‌های می‌توان به ایجاد محیط‌های منزوی در هر پروژه اشاره کرد که خطاهای مربوط به محیط توسعه را کاهش می‌دهد.

# ۴ فصل چهارم تکنولوژی‌های استفاده شده تکنولوژی‌های استفاده شده

در این بخش به توضیح تکنولوژی‌های استفاده شده در این پروژه می‌پردازیم. از آنجایی که بخش‌های عمده‌ای از این پروژه را تکنولوژی‌های بیرونی تشکیل می‌دهند توضیح آن‌ها از اهمیت بالایی برخوردار است.



## ابزار پرومتئوس[[18]](#footnote-18)

‌پرومتئوس یک پروژه‌ی نهاد کلود نیتیو کامپیوتینگ[[19]](#footnote-19) می‌باشد که برای مبصری سرویس‌ها و سیستم‌ها استفاده می‌شود. پرومتئوس شاخص‌ها را از هدف‌ها در بازه‌های زمانی داده شده جمع‌آوری می‌کند، قوانین تعریف شده را محاسبه می‌کند و نتایج را نشان می‌دهد. هم‌چنین توانایی آن را دارد که هشدارهایی را در صورت لزوم فعال نماید اگر شرط از پیش تعریف شده‌ای رخ دهد.

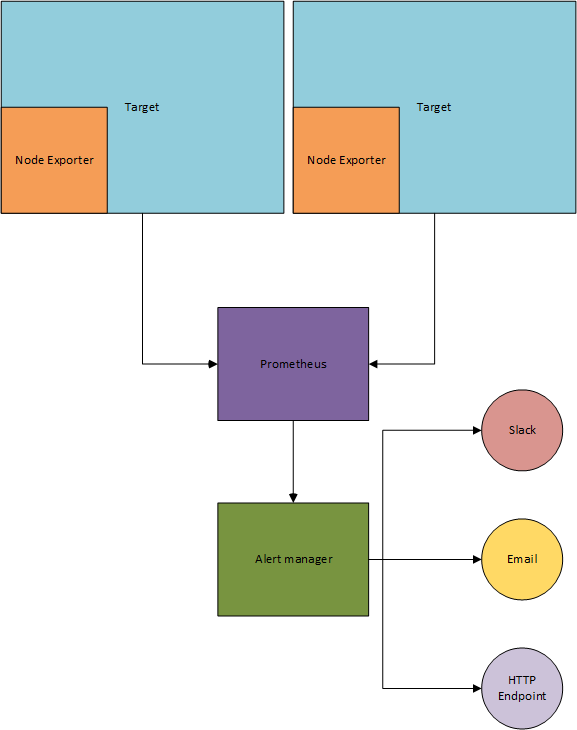


شکل ‏0‑1 - لوگوی ابزار پرومتئوس

معماری پرومتئوس به این صورت است که می‌توان روی سیستم‌های هدف نرم‌افزارهای موردنیاز برای گزارش نصب کرد. به این نرم‌افزار اصطلاحا اکسپورتر[[20]](#footnote-20) گفته می‌شود. اکسپورتر به این صورت هستند که داده‌های جمع‌آوری شده خود را در قالب خاصی گزارش می‌کنند. گزارش کردن آن‌ها نیز به این صورت است که در یک مسیر مشخص توسط پروتکل اچ‌تی‌تی‌پی قرار داده می‌شود و سرور مرکزی پرومتئوس در بازه‌های زمانی مشخص، این اطلاعات را جمع‌آوری می‌کند.

### سیستم ایجاد هشدار در پرومتئوس

در شکل زیر شما می‌توانید مکانیزم ایجاد هشدار در پرومتئوس را مشاهده کنید.



شکل ‏0‑2 - نحوه‌ی کارکرد مکانیزم هشدار در پرومتئوس

مکانیزم ایجاد هشدار در پرمتئوس به این صورت است که شما قوانین هشدار را در داخل پرمتئوس تعریف می‌کنید و سپس آدرس برنامه مدیریت هشدار را به برنامه پرومتئوس می‌دهید. این برنامه یک برنامه جداگانه و مستقل از برنامه اصلی پرومتئوس می‌باشد. پس تعریف هشدارها پرومتئوس خود دائما بررسی می‌کند که قوانین هشدار تعریف شده در چه زمانی برقرار می‌شوند. پس از برقراری اطلاعات هشدار را برای برنامه مدیریت هشدار ارسال می‌کند. برنامه مدیریت هشدار پس از انجام مجموعه‌ای از شروط اولیه (مانند تکراری نبودن هشدار در زمان‌های اخیر) آن را در مسیر تعریف شده قرار می‌دهد. به صورت پیش‌فرض سرویس‌های بسیار زیادی را پشتیبانی می‌کند. با این حال اگر سرویس شما در بین آن سرویس‌ها نباشد به شما این امکان را می‌دهد تا یک مسیر و آدرس را مشخص کنید و سپس هشداردهنده اطلاعات مربوطه را با فعل اچ‌تی‌تی‌پی پست[[21]](#footnote-21) برای شما ارسال خواهد کرد. حال انجام هر عملیاتی برعهده شما می‌باشد.

## ابزار انسیبل[[22]](#footnote-22)

انسیبل نرم‌افزار متن‌بازی است که پیکبربندی نرم‌افزار را به صورت اتوماتیک انجام می‌دهد. انسیبل از طریق اس‌اس‌اچ عملیات پیکربندی را انجام می‌دهد. انسیبل عملیات پیکربندی را از طریق یک نود مدیریت‌کننده انجام می‌دهد. این نود لیستی از سیستم‌های هدف را در اختیار دارد که وظیفه دارد برروی آن‌ها عملیات پیکربندی را انجام دهد. سپس عملیات‌های تعیین شده را روی سیستم‌های مقصد انجام می‌دهد.

انسیبل در هنگامی که در حال اجرا نیست هیچ منبعی مصرف نمی‌کند. چون انسیبل هیچ برنامه‌ای ندارد که در پشت زمینه در حال اجرا باشد. یکی از بخش‌هایی که انسیبل را بسیار قدرتمند می‌سازد ماژول‌ها می‌باشند. هدف ماژول‌ها این است که حتی اگر یک عملیات را چندبار انجام دهند بازهم سیستم را در یک حالت موردنظر قرار دهند. در صورتی که شرط موردنظر برقرار نباشد ممکن است منجر به محیط‌هایی با حالات مختلف گردد.

در این پروژه ما از انسیبل برای پیکربندی اولیه سیستم استفاده کرده‌ایم. یعنی پس از بالا آمدن ماشین مجازی عملیات نصب داکر و اکسپورتر از این طریق صورت می‌پذیرد.



شکل ‏0‑3 - لوگوی انسیبل

## محیط‌تست ساوی[[23]](#footnote-23)

ساوی یک تست بِد با شش منطقه در سراسر کانادا می‌باشد که امکان تست زیرساخت‌های مبتنی بر شبکه جدید را فراهم می‌سازد. یکی از خدماتی که ساوی ارائه می‌دهد زیرساخت به عنوان سرویس می‌باشد. به این معنی که شما می‌توانید از ساوی درخواست ماشین مجازی کنید. تنها استفاده‌ی ما از ساوی در این پروژه همین می‌باشد.

## زبان برنامه‌نویسی پایتون[[24]](#footnote-24)

پایتون یک زبان برنامه‌نویسی مفسری سطح بالا می‌باشد که برای برنامه‌نویسی عام‌منظوره استفاده می‌شود. فلسفه پایتون بر خوانا بودن کد است که این فلسفه را از طریق استفاده از فاصله به اندازه کافی فراهم می‌آورد. پایتون ساختاری را فراهم می‌آورد که هم در پروژه‌های کوچک و هم پروژه‌های بزرگ خوانایی کد بالا باشد.

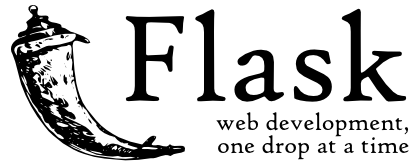
ما در این پروژه به چندین دلیل از پایتون استفاده کردیم. دلیل اول استفاده از آن این است که انسیبل تنها کتابخانه‌ی موجود برای کار با آن به زبان پایتون موجود بود. دلیل دیگر استفاده از پایتون آن است که کتابخانه‌های بسیار غنی‌ای در رابطه با حوزه‌ی کاری این پروژه وجود دارد. به عنوان مثال تنها کتابخانه رسمی داکر به زبان پایتون می‌باشد.

C:\Users\iman\Documents\University\BSc Project\Python_logo_and_wordmark.svg.png

شکل ‏0‑4 - لوگوی زبان برنامه‌نویسی پایتون

## کتابخانه فلسک[[25]](#footnote-25)

از این کتابخانه برای ایجاد نقطه‌پایانی مربوط استفاده شده است. فلسک یک ریزچارچوب برای ایجاد نقاط پایانی اچ‌تی‌تی‌پی می‌باشد. از اهداف پیاده‌سازی این کتابخانه می‌توان به هسته‌ی ساده آن اشاره کرد. در این کتابخانه هسته بسیار کوچک می‌باشد ولی قابل گسترش است. از دیگر اهداف طراحی این کتابخانه می‌توان به این موضوع اشاره کرد که فلسک هیچ تصمیمی را از پیش برای شما نمی‌گیرد و تمامی تصمیم‌ها با شما است. این بدین معناست که به عنوان مثال اینکه قرار است پایانه‌های اچ‌تی‌تی‌پی شما داده‌ها را چگونه ذخیره کنند هیچ پیش‌فرضی را در نظر نگرفته است و تمامی آن برعهده شما می‌باشد.



شکل ‏0‑5 - لوگوی کتابخانه فلسک

## ردیس[[26]](#footnote-26)

ردیس یک دیتابیس داخل حافظه‌ی متن‌باز می‌باشد که می‌توان اغلب داده‌ساختارها را در آن ذخیره کرد. از ردیس به عنوان پایگاه‌داده، حافظه‌ی کش و مبادله‌کننده پیام استفاده می‌شود. از داده‌ساختارهایی که ردیس از آن استفاده می‌کند می‌توان به رشته‌ها، هش‌[[27]](#footnote-27)ها، لیست‌ها، مجموعه‌ها، مجموعه‌های مرتب شده و بسیاری از داده‌ساختارهای دیگر اشاره کرد. ردیس به صورت پیش‌فرض از کپی کردن، اسکریپت‌نویسی لوآ، ذخیره‌سازی داده‌های داخل حافظه برروی دیسک و فراهم‌آوری در دسترس‌پذیری بالا را به صورت پیش‌فرض در خود دارد.

در این پروژه از ردیس به عنوان ذخیره‌ساز صف استفاده شده است. چون در اینجا برخی از تقاضاها زمان زیادی را مصرف می‌کنند، استفاده از آن باعث می‌شود تا زمان معطل کردن هر تقاضا کاهش پیدا کند و تقاضای مربوطه به داخل صف اضافه شود.



شکل ‏0‑6 - لوگوی ردیس

## کتابخانه سِلِری[[28]](#footnote-28)

سِلِری یک کتابخانه در زبان برنامه‌نویسی پایتون است که ویژگی‌های زیر را فراهم می‌کند.

* سادگی
* در دسترس‌پذیری بالا
* سریع بودن
* انعطاف پذیر

کلری می‌تواند از صف‌های پیام متعددی استفاده کند. از مثال‌های این صف‌های پیام می‌توان به موارد زیر اشاره کرد:

* ردیس
* ربیت‌‌ام‌کیو
* آمازون اس‌کیو‌اس

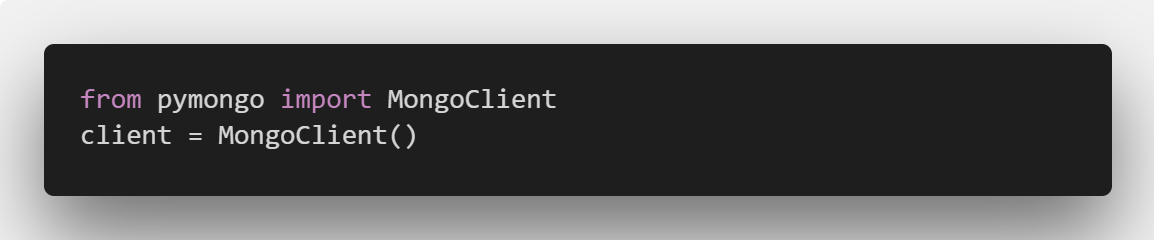
از دیگر ویژگی‌های کلری می‌توان به آسانی ادغام شدن با چارچوب‌های مختلف اشاره می‌کرد. کلری می‌تواند به آسانی با فلسک ادغام می‌شود. نمونه تعریف یک وظیفه‌ای که به صورت غیرهمگام اجرا می‌شود را می‌توانید در بخش زیر مشاهده کنید:



شکل ‏0‑7 - قطعه کد تعریف عملیات ناهمگام در کلری

## کتابخانه پای‌مانگو[[29]](#footnote-29)

پای‌مانگو یک کتابخانه برای کار با دیتابیس مانگو می‌باشد. پای‌مانگو روش توصیه شده کار با دیتابیس مانگو در پایتون می‌باشد. از ویژگی‌های این کتابخانه می‌توان به رابط برنامه‌نویسی بسیار ساده اشاره کرد. برای اتصال به پایگاه داده در پای‌مانگو قطعه کد زیر کافی می‌باشد. ]۳[



شکل ‏0‑8- نحوه استفاده از کتابخانه پای‌مانگو

## دیتابیس مانگودی‌بی[[30]](#footnote-30)

مانگو یک دیتابیس رایگان و متن‌باز می‌باشد که بر مبنای سند‌ها توسعه داده شده است. مانگو یک دیتبایس نو‌اس‌کیو‌ال محسوب می‌شود. مانگودی‌بی از اسناد شبیه جی‌سان استفاده می‌کند. مانگودی‌بی توسط شرکت مانگودی‌بی توسعه داده شده است. از ویژگی‌های مانگو‌دی‌بی عدم تاکید بر ساختارمند بودن داده قراردادن شروط سفت و سخت برروی مدل‌ها می‌توان اشاره کرد. در شکل زیر می‌توانید یک نمونه از داکیومنت را در مانگو‌دی‌بی مشاهده کنید:



شکل ‏0‑9 - نمایش یک سند نمونه در مانگو‌دی‌بی

# ۵ فصل پنجم نحوه عملکرد سیستم‌توسعه‌پذیرکننده

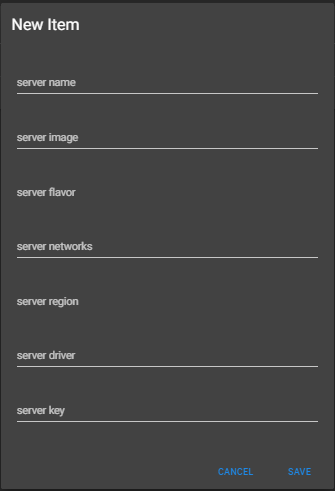
# نحوه عملکرد سیستم‌توسعه‌پذیرکننده

در این بخش قصد داریم تا درباره‌ی نحوه‌ی عملکرد سیستم توسعه‌پذیرکننده صحبت کنیم. این فصل اولین فصلی می‌باشد که درباره ساختار داخلی و نحوه‌ی عملکرد آن توضیح می‌دهد. دلیل به تاخیر افتادن این فصل نیازمندی توضیح تکنولوژي‌های استفاده شده در سیستم می‌باشد. برای توضیح نحوه‌ی عملکرد سیستم در اینجا در ابتدا نحوه کارکرد آن را توضیح می‌دهیم و سپس رفتار سیستم را در مقابل رویدادها توضیح می‌دهیم. هم‌چنین در این بخش نحوه‌ی کار با بخش‌های مختلف سیستم نیز توضیح داده می‌شود.



## نحوه‌ی ساخت ماشین مجازی

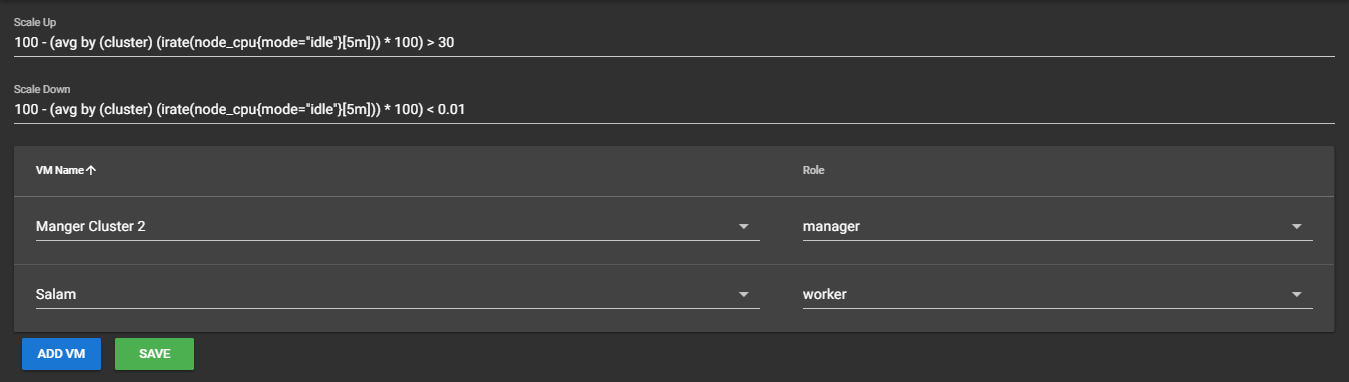
در حال حاضر تنها از فراهم‌کننده‌ی ابری اپن‌استک پشتیبانی می‌شود. نحوه ساختن ماشین مجازی به این صورت است که شما می‌توانید تمامی پارامترهای مربوط به ساخت ماشین مجازی را مشخص کنید و سپس دستور ساخت آن را صادر نمایید. پس از ارسال تقاضا درخواست شما درون یک صف قرار داده می‌شود و پس از اتمام ساخت شما می‌توانید ماشین مجازی خود را در لیست ماشین‌های مجازی مشاهده نمایید. از کارهایی که درون صف انجام می‌شود می‌توان به نصب داکر، پایتون و اکسپورتر و هم‌چنین پیکربندی پورت‌ها برای آمادگی کار با سیستم را داشتن اشاره کرد. در حال حاضر عمل ساختن ماشین مجازی هم از طریق رابط کاربری و هم رابط برنامه‌نویسی قابل انجام است.



شکل ‏0‑1 - ایجاد سرور جدید

## نحوه تعریف خوشه‌های کانتینری داکر سوارم

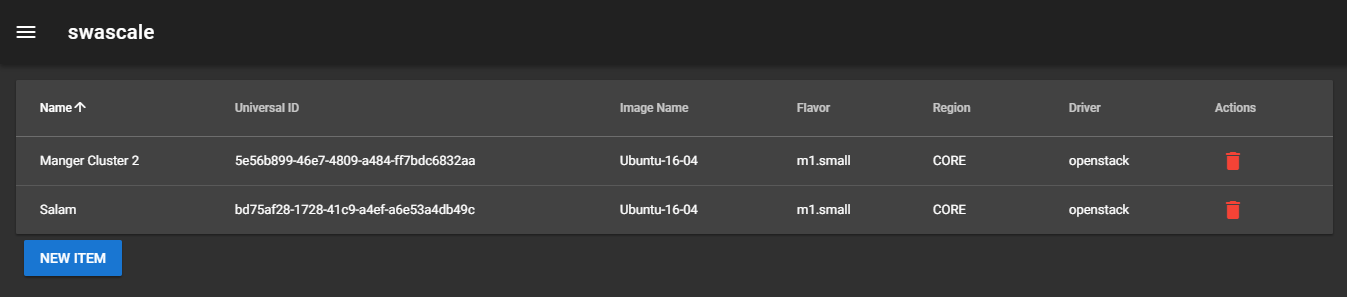
سیستم به شما این امکان را می‌دهد که خوشه‌های کانتینری را تعریف کنید و در آن نقش هر یک از گره‌ها را تعیین نمایید. تنها کامپیوترهایی توانایی پیوستن به خوشه را دارند که توسط روش توضیح داده شده در بخش ۱-۵ ایجاد شده باشند. از طریق رابط کاربری و رابط برنامه‌نویسی می‌توان خوشه‌های متعدد کانتینری تعریف کرد که در هریک از آن‌ها نقش هر سرور مشخص شده باشد. این خوشه‌هایی که تعریف می‌شوند توانایی توسعه‌ی خودکار به سمت بالا یا پایین را دارند. یعنی شما می‌توانید هر پارامتری را به عنوان پارامتری که قرار است براساس آن به سمت بالا یا پایین توسعه‌ی خودکار صورت گیرد را مشخص کنید.



شکل ‏0‑2 - تعریف خوشه‌ی کانتینری جدید

## مشاهده‌ی ماشین‌های مجازی موجود

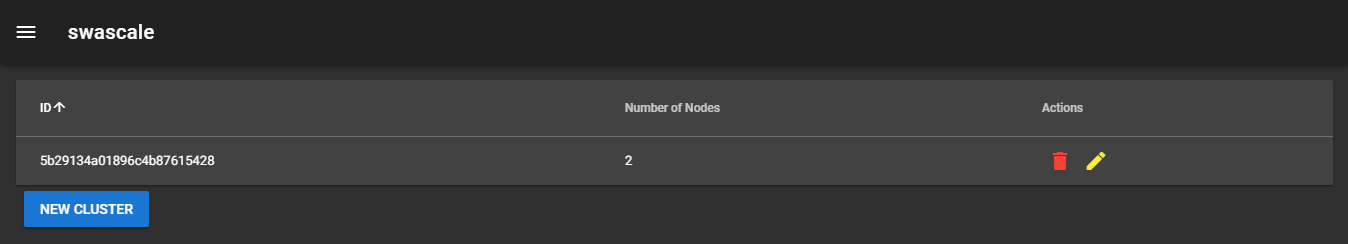
شما می‌توانید تمامی ماشین‌های مجازی‌ای که تعریف کرده‌اید را مشاهده نمایید. تصویر زیر بخشی از تصویر مشاهده‌ی ماشین‌های مجازی موجود می‌باشد.



شکل ‏0‑3 - مشاهده‌ی ماشین‌های مجازی موجود

## مشاهده‌ی خوشه‌های تعریف شده

شما می‌توانید تمامی خوشه‌های کامپیوتری‌ای را که تعریف کرده‌اید، مشاهده نمایید. تصویر زیر بخشی از تعریف خوشه‌های کامپیوتری می‌باشد.

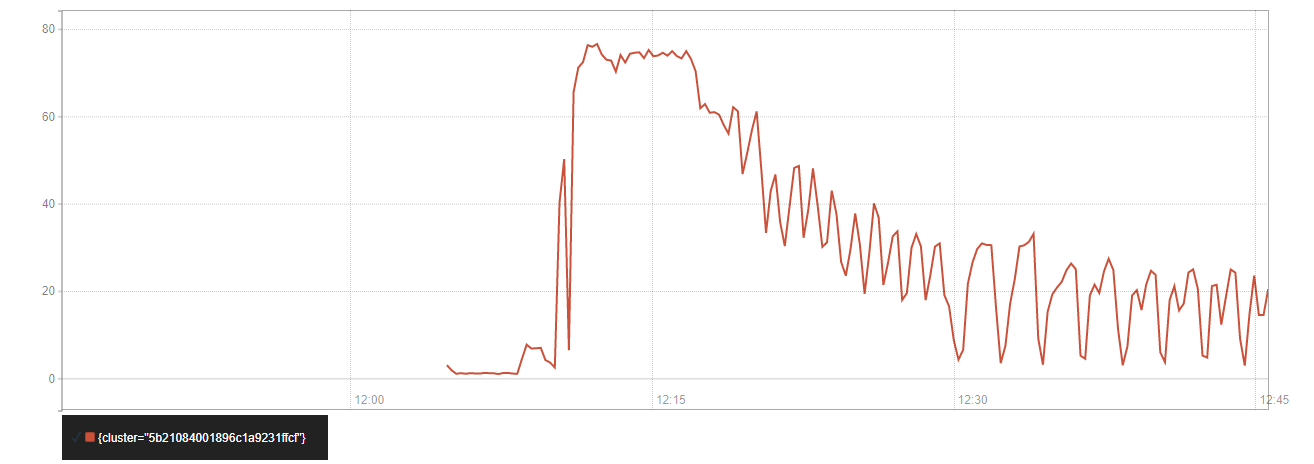


شکل ‏0‑4- مشاهده‌ی خوشه‌های کانتینری تعریف شده

## سناریو اول

در این بخش قصد داریم عملکرد سیستم را در مقابل یک سناریو شرح دهیم. این سناریو به صورت زیر می‌باشد. یک خوشه‌ی کامپیوتری با دو کامپیوتر تعریف کرده ایم. یکی از این گره‌ها به عنوان مدیر خوشه و دیگری به عنوان کارگر می‌باشد. با استفاده از نرم‌افزار اچ‌تی‌تی‌پی پرف که برای ایجاد تقاضا در سطح بالا می‌باشد، به صورت تقریبی در هر ثانیه ۷۰۰ تقاضا ایجاد می‌نماییم. این حجم از تقاضا باعث بالارفتن میزان مصرف قدرت پردازشی خوشه کامپیوتری می‌گردد. ما برای خوشه‌ی کامپیوتری شرط بالای ۳۰ درصد را برای توسعه به سمت بالا تعریف کرده‌ایم. این یعنی در صورتی که میزان مصرف پردازشی بیش‌تر از ۳۰ درصد باشد، سیستم به صورت خودکار یک سرور به خوشه به عنوان کارگر اضافه می‌کند.

این سناریو را به صورت واقعی اجرا کردیم و در شکل زیر نمودار مصرف پردازشی خوشه به صورت میانگین را می‌توانید مشاهده نمایید.

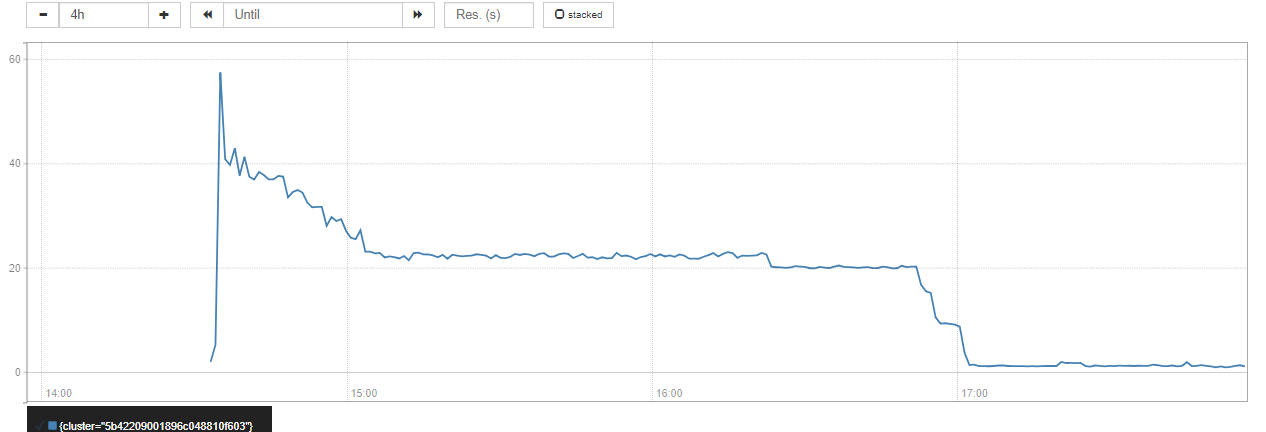


شکل ‏0‑5 - واکنش سیستم به ۷۰۰ تقاضا در ثانیه

همانطور که در نمودار مشاهده می‌کنید در ابتدا میزان مصرف پردازشی به صورت ناگهانی افزایش پیدا کرده است. اما پس از اضافه کردن ۷ گره به عنوان کارگر بار بین آن‌ها تقسیم شده و در نهایت سیستم به وضعیت پایدار با کمتر ۳۰ درصد مصرف پردازشی میانگین رسیده است.

## سناریو دوم

در این سناریو ما از ۵ گره استفاده کردیم. از این گره‌ها دوتای آن‌ها گره مدیر می‌باشند و سه تای آن‌ها گره کارگر می‌باشند. پارامترهای توسعه‌ی خودکار به سمت بالا را برابر با مصرف سی‌پی‌یو بیش از ۳۰ درصد قرار دادیم و پارامتر توسعه‌ی خودکار به سمت پایین را برابر با مصرف سی‌پی‌یو کمتر از ۳ درصد. برای آن‌که مصرف سی‌پی‌یو این مجموعه‌ی گره‌ها را به بیش از ۳۰ درصد برسانیم لازم بود تا از دو ماشین مجازی برای این منظور استفاده کنیم. هر کدام از این ماشین‌ها به صورت میانگین ۱۴۰۰ تقاضا بر ثانیه بر گره مدیر وارد می‌کردند. پس از وارد کردن این حجم از تقاضا میزان مصرف سی‌پی‌یو در خوشه به صورت میانگین بیش از ۳۰ درصد شد. پس از به ثبات رسیدن سیستم با اضافه کردن ۲ گره ما بار سیستم را صفر رساندیم تا فعالیت کاهش گره‌ها و توسعه‌پذیری به سمت پایین را مشاهده کنیم. در شکل زیر می‌توانید واکنش سیستم به این بار را مشاهده کنید.



شکل ‏0‑6 – واکنش سیستم به سناریو شماره دوم

در شکل فوق در قسمت‌هایی که بار افزایش پیدا کرده است مصرف سی‌پی‌یو برای بازه‌هایی بیش از ۳۰ درصد شده است و سپس پس از برداشتن بارها به ترتیب مصرف سی‌پی‌یو به کمتر از سه درصد رسیده است که باعث حذف گره‌ها شده است و با تنها یک گره مدیر به وضعیت پایانی رسیده است.

# ۶ فصل ششم جمع‌بندي و نتيجه‌گيري و پیشنهاداتجمع‌بندي و نتيجه‌گيري

همانطور که در این پروژه دیدیم، توسعه پذیری خودکار در صورت پیاده‌سازی صحیح و دقیق می‌تواند روش بسیار مفید و مقرون به صرفه‌ای باشد. از آن‌جایی توسعه‌ی خودکار نیاز به نیروهای تیم عملیات را به شدت کاهش می‌دهد. هم‌چنین دسترس‌پذیری سرویس را نیز به طور قابل ملاحظه‌ای افزایش می‌دهد. به همین جهت استفاده از این سیستم‌ها برای پروژه‌های بزرگ اجتناب ناپذیر خواهد بود.

**پیشنهادات**

همانطور که در فصل دوم درباره‌ی انواع توسعه‌پذیری صحبت شد، یکی دیگر از روش‌های توسعه‌پذیری روش توسعه‌پذیری پیش‌گوینده می‌باشد. در این روش با مشاهده‌ی روندهای اطلاعات در گذشته و پیش‌بینی بار قبل از وقوع حادثه نسبت به انجام توسعه‌پذیری اقدام می‌نماید. این روش، در صورت پیش‌بینی صحیح روش کارآمدتری از روش توسعه‌پذیری مطرح شده در این پروژه می‌باشد. هم‌چنین می‌توان این پیش‌بینی‌ها را براساس الگوریتم‌های یادگیری ماشین و داده‌های حجیم انجام داد.

بخش دیگر قابل ادامه دادن در این پروژه کارشناسی، تحقیق بر روی پارامترهایی که توسعه‌پذیری را بهینه می‌کنند، می‌باشد. از آنجایی که دست محقق در این پروژه کاملا برای انتخاب پارامتر موردنظر باز است و می‌تواند تاثیر انتخاب پارامتر را برروی سیستم به صورت لحظه‌ای مشاهده کند، به این منظور از این پروژه می‌توان در این راستا نیز استفاده کرد.

استفاده از روش‌های اضافه کردن ماشین‌ها به صورت نمایی به جای دانه دانه، به منظور پاسخ‌گویی سریع‌تر به نیازمندی‌ها نیز از بخش‌های دیگر که می‌تواند به پروژه اضافه شود، می‌توان اشاره کرد.

# منابع و مراجع

[1] P Mell, T. G. (2011). The NIST definition of cloud computing. *Recommendations of the National Institute*, 1-3.

[2] *Production-Grade Container Orchestration - Kubernetes*. (2018, June 23). Retrieved from Kubernetes: https://kubernetes.io/

[3] *PyMongo 3.6.1 Documentation*. (2018). Retrieved from https://api.mongodb.com/python/current/

[4] Roger S. Pressman, B. M. (2014). *Software Engineering: A Practitioner's Approach.* McGraw-Hill Education.

[5] *Swarm mode key concepts*. (2018, June 23). Retrieved from https://docs.docker.com/engine/swarm/key-concepts/

**Abstract**

With the rise of internet technologies and the dynamic number of users of these services, systems that can adapt themselves with the number of users have found significant application. In addition, with the widespread use of containers, systems that can automatically adjust container clusters have also found crucial importance. In this project, we will implement a system that provides auto scaling for container clusters.

**Key Words:** auto-scaling, Docker, Cloud Computing



Amirkabir University of Technology  
(Tehran Polytechnic)

Computer Engineering and Information Technology

B.Sc. Project

An auto scaler for Docker Swarm

By

Iman Tabrizian

Supervisor

Dr. Mahdi Dehghan

June 2018

1. Usability [↑](#footnote-ref-1)
2. Usability Professionals Association [↑](#footnote-ref-2)
3. Isolated [↑](#footnote-ref-3)
4. Cloud Orchestrators [↑](#footnote-ref-4)
5. Scheduling [↑](#footnote-ref-5)
6. Container Orchestrator [↑](#footnote-ref-6)
7. Redis Cluster [↑](#footnote-ref-7)
8. Docker Swarm [↑](#footnote-ref-8)
9. Kubernetes [↑](#footnote-ref-9)
10. Load Balancing [↑](#footnote-ref-10)
11. Scryer [↑](#footnote-ref-11)
12. Declarative [↑](#footnote-ref-12)
13. Raft [↑](#footnote-ref-13)
14. Usecase [↑](#footnote-ref-14)
15. OpenStack [↑](#footnote-ref-15)
16. VS Code [↑](#footnote-ref-16)
17. Pipenv [↑](#footnote-ref-17)
18. Prometheus [↑](#footnote-ref-18)
19. Cloud Native Computing Foundation [↑](#footnote-ref-19)
20. Exporter [↑](#footnote-ref-20)
21. HTTP POST [↑](#footnote-ref-21)
22. Ansible [↑](#footnote-ref-22)
23. SAVI Testbed [↑](#footnote-ref-23)
24. Python [↑](#footnote-ref-24)
25. Flask [↑](#footnote-ref-25)
26. Redis [↑](#footnote-ref-26)
27. Hash [↑](#footnote-ref-27)
28. Celery [↑](#footnote-ref-28)
29. Pymongo [↑](#footnote-ref-29)
30. MongoDB [↑](#footnote-ref-30)