دانشگاه شهید بهشتی

دانشکده مهندسی و علوم کامپیوتر

کاهش شروع‌های سرد در پلتفرم‌های بدون سرور

گزارش سمینار کارشناسی ارشد مهندسی کامپیوتر
گرایش نرم‌افزار

نگارش

امیرمحمد کرم‌زاده

استاد راهنما

دکتر علیرضا شاملی

خرداد ۱۴۰۰

چکیده

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ و با استفاده از طراحان گرافیک است. چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است و برای شرایط فعلی تکنولوژی مورد نیاز و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد. کتابهای زیادی در شصت و سه درصد گذشته، حال و آینده شناخت فراوان جامعه و متخصصان را می طلبد تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی و فرهنگ پیشرو در زبان فارسی ایجاد کرد. در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها و شرایط سخت تایپ به پایان رسد وزمان مورد نیاز شامل حروفچینی دستاوردهای اصلی و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

واژگان کلیدی: رایانش ابری، رایانش بدون سرور، شروع‌های سرد، function-as-a-service، FaaS

فهرست مطالب

۱ مقدمه **۱**

۱.۱ صورت مسئله ۲

۲.۱ انگیزه‌ی تحقیق ۲

۳.۱ مثال مرتبط ۳

۴.۱ اهمیت موضوع ۴

۵.۱ نتیجه‌های مهم تحقیق ۵

۲ مروری بر ادبیات **۶**

۱.۲ رایانش بدون سرور ۷

۱.۱.۲ تعریف رایانش بدون سرور ۷

۲.۱.۲ معماری ۱۰

۳.۱.۲ ویژگی‌های پلتفرم‌های بدون سرور ۱۱

۴.۱.۲ پلتفرم‌های تجاری ۱۴

۵.۱.۲ پلتفرم‌های آزاد و متن باز ۱۷

۳ کارهای مرتبط **۱۸**

۴ نتیجه‌گیری **۲۰**

فهرست شکل‌ها

۱.۱ اهمیت شروع‌سرد ۴

۱.۲ مرزهای رایانش بدون سرور و رایانش سرورآگاهانه ۹

۲.۲ معماری کلی یک پلتفرم بدون سرور ۱۰

فهرست جداول

فصل ۱

مقدمه

۱.۱ صورت مسئله

چه روش‌هایی برای کاهش تعداد شروع‌های سرد در پلتفرم‌های بدون‌سرور [۱] با حداقل سربار [۲] و زمان اجرایی وجود دارند؟

۲.۱ انگیزه‌ی تحقیق

رایانش بدون سرور[۳] یکی از مسائل داغ و محبوب این‌روزهای دنیای مهندسی نرم‌افزار و رایانش ابری است. رایانش بدون سرور حوزه‌ی جدیدی را در توسعه‌ی محصول و استقرار[۴] اپلیکیشن‌ها باز کرده است. یکی از دلایل محبوبیت استفاده از پلتفرم‌های بدون‌سرور و تمایل توسعه‌دهندگان برای مهاجرت به سمت آن، استفاده بیش از پیش از معماری میکروسرویس و نانوسرویس در توسعه‌ی محصولات و حرکت معماران و مهندسین نرم‌افزار در تولید و مهاجرت برنامه‌های کاربردی با این معماری‌ها است.

از دید توسعه دهنده، رایانش ابری با حذف دخالت مستقیم کاربران انتهایی در مدیریت زیرساخت ازجمله lrauto-scaling یا lrload-balancing موجب بهبود سرعت توسعه محصول و تمرکز کاربران برروی منطق[۵] برنامه است. همچنین، برای علاوه‌بر آسانی استفاده و پنهان‌سازی پیچیدگی مدیریت سرور از کاربر، به علت اینکه ارائه‌دهندگان خدمات ابری در نقاط مختلف جهان حضور دارند و همچنین کانفیگ بهینه CDNها؛ ارتباطات بین سرورها و کاربران با حداقل تاخیر[۶] صورت می‌گیرد.

به طور کلی، یک پلتفرم بدون‌سرور را هر پلتفرم محاسباتی تعریف کرد که در آن مدیریت مستقیم سرور از کاربران مخفی شده و برنامه‌های کاربردی به صورت اتوماتیک در آن مقیاس‌پذیر می‌شوند و تنها هنگامی که در حال استفاده از پلتفرم هستیم، هزینه آن را پرداخت می‌کنیم. [۱]

یکی از قابلیت‌هایی که در رایانش بدون سرور باعث محبوبیت آن شده‌است، قابلیت Scale-to-Zero است. این بدان معنی است که هنگامی که از یک کانتینر استفاده‌ای نداریم، منابع آن گرفته می‌شوند و کانتینر اصطلاحا

[1] serverless
[2] Overhead
[3] Serverless Computing
[4] Deployment
[5] logic
[6] Latency

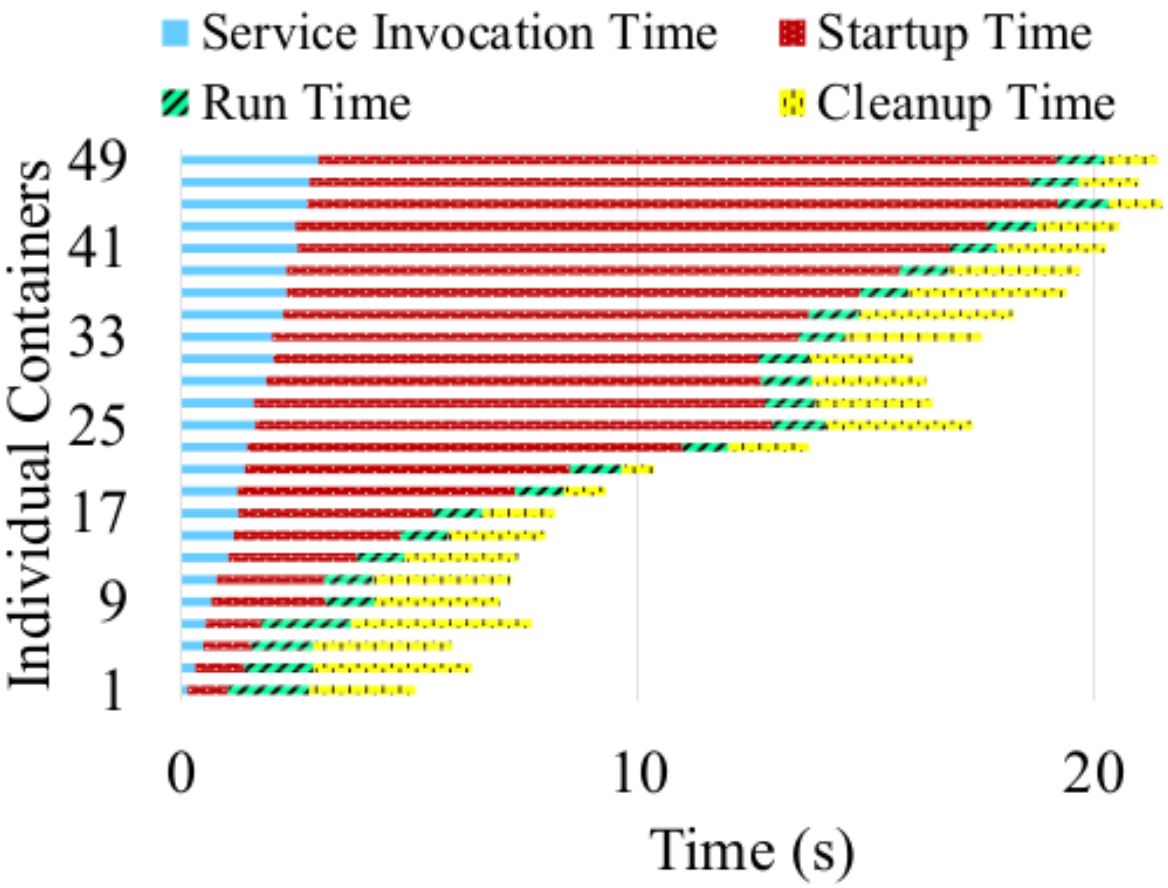
Zero-Scaled می‌شود. این خود موجب قابلیت پرداخت تنها در حین مصرف ما از تابع می‌شود. اما مشکل اصلی زمانی است که درخواست جدیدی برای کانینر Zero-Scale شده می‌رسد؛ در این حالت باید درخواست منتظر مانده تا سلسله‌ای از آماده‌سازی ها انجام شوند تا کانتینر مربوطه مجددا اجرا شود. این خود باعث تاخیری مضاف برای پاسخ‌دهی به درخواست را موجب می‌شود که به این تاخیر مشکل شروع‌سرد [1] گفته می‌شود. در واقع می‌توان گفت تاخیر شروع سرد ناشی از تلاش ما در تعادل بین تاخیر در پاسخگویی به درخواست‌ها و هزینه (هزینه‌های استفاده از رم و سی‌پی‌یو و ...) است.

متاسفانه در سالیان اخیر و در عین داغ‌بودن مبحث و نیاز بازار به حل این مشکل، این مشکل چندان در محیط‌های آکادمیک مورد بررسی قرار نگرفته است. البته در نگاه کلی‌تر، مشکلات و مسائل بار مربوط به رایانش بدون‌سرور، اکثرا در محیط‌های آکادمیکی مثل دانشگاه‌ها با کم محلی روبرو شده‌اند. در این میان، پلتفرم‌های متن‌باز [2] که دارای جوامع بسیار گسترده‌ای نیز هستند، به خاطر این سری مسائل باز که شرکت‌های تجاری در حال صرف هزینه‌های هنگفتی برای حل و فصل مسکلات مربوط به آن هستند، به شدت از رقابت عقب مانده‌اند. انگیزه ما برای انجام این پژوهش در این است که اولا بتوانیم به راهکار مناسب‌تری برای حل مشکل مربوط به شروع سرد در پلتفرم‌های بدون سرور برسیم و ثانیا بتوانیم با مشارکت در بهبود یکی از پلتفرم‌های آزاد در توسعه این پلتفرم‌ها تاثیر کوچکی داشته باشیم.

۳.۱ مثال مرتبط

در مقاله [۲] آقای لین و همکارش توانستند تا با اسفاده از استخر گرم و نگه‌داری کانتینر توابعی که محبوبیت استفاده دارند، مدت زمان پاسخ را در حدود ٪۸۵ کاهش دهند. این بهبود در پلتفرم knative اجرا شد و ایده‌ی مقالات دیگری نیز بوده است.

[1] Cold Start
[2] Open Source



شکل ۱.۱: اهمیت شروع‌سرد

۴.۱ اهمیت موضوع

اگرچه پلتفرم‌های بدون‌سرور از نظر هزینه، scaling، راحتی استفاده و گستردگی پوشش جغرافیایی برای ما بهینه هستند؛ اما مشکل شروع سرد مشکلی نیست که بتوان به سادگی از آن گذر کرد. تصویر ۱.۱ که از [۳] گرفته شده است، نشان می‌دهد که بیش از ٪۸۰ زمان اجرای کامل یک کانتینر در پلتفرم‌های بدون سرور به آماده‌سازی آن یا شروع سرد اولیه[1] مربوط می‌شود.

ستون قرمز رنگ زمان آماده سازی کانتینر را نمایش می‌دهد. این زمان همان زمان شروع سرد است که دلایل مختلفی از جمله آماده‌سازی کانتینر یا حضور در صف انتظار برای تخصیص منابع باشد. بنابراین، با به‌کارگیری یک استراتژی مناسب می‌توان این زمان را به حداقل رسانید.

متاسفانه تاخیر شروع سرد باعث شده تا توسعه‌دهندگان اقبال کمتری به استفاده از پلتفرم‌های بدون سرور

[1] First Cold Start

داشته باشند. به گونه‌ای که از بین ۱۰۰۰ برنامه‌ی کاربردی بزرگ در پلتفرم Microsoft Azure، تنها یک مورد مربوط به یک برنامه تجاری باشد [۴]. این موضوع نشان می‌دهد که علی‌رغم پتانسیل بالای رایانش بدون سرور، وجود مشکلات جدی ازجمله شروع سرد، باعث امتناع توسعه‌دهندگان از مهاجرت به پلتفرم‌های بدون سرور باشد.

۵.۱ نتیجه‌های مهم تحقیق

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

ساختار این گزارش به این ترتیب خواهد بود:

درادبیات موضوع مروری بر واژگان، مفاهیم تخصصی و هر آن‌چه که در ادامه به آن نیاز پیدا خواهیم کرد، خواهیم داشت. سپس در فصل کارهای مرتبط به بیان مشروح تحقیقات انجام شده خواهیم پرداخت و در نتیجه‌گیری و کارهای آینده خلاصه‌ای از مطالعات انجام شده و مسائل باز خواهیم پرداخت.

فصل ۲

مروری بر ادبیات

در این بخش سعی داریم تا با مروری بر اصطلاحات و ابزارهای مورد استفاده در پژوهش‌های بررسی شده، با پیش‌نیازهای مبحث موردنظر آشنا شویم.

۱.۲ رایانش بدون سرور

رایانش بدون سرور[1] در سال ۲۰۱۴ توسط شرکت آمازون برای اولین بار معرفی شد. تا قبل از این رایانش بدون سرور یک مفهوم انتزاعی [2] در شبکه بود که شرکت آمازون با ارائه پلتفرم AWS Lambda Functions [؟] به معرفی آن پرداخت. سپس در سال ۲۰۱۶ سایر ارائه دهندگان خدمات ابری نیز به ارائه پلتفرم‌های بدون سرور خود پرداختند. در این سال به ترتیب شرکت‌های گوگل پلتفرم google cloud functions یا به اختصار GCP، شرکت مایکروسافت پلتفرم Microsoft Azure functions و شرکت IBM به معرفی IBM OpenWhisk پرداختند. البته باید توجه داشت که مفهوم رایانش بدون سرور به طور کامل توسط ارائه دهندگاه خدمات ابری پیاده‌سازی نشده است و جای کار بسیاری دارد (با مطالعه این گزارش به مرور متوجه نواقص موجود خواهید شد).

در رایانش بدون سرور ما از نقطه قوت ماشین‌های مجازی که ایزولاسیون برنامه‌های مختلف از همدیگر بود استفاده کرده‌ایم. منتها این مورد را با مفهوم کانتینر ها پیاده‌کرده ایم. در ادامه راجع به کانتینرها نیز بحث خواهیم کرد.

۱.۱.۲ تعریف رایانش بدون سرور

رایانش بدون سرور مبحثی از رایانش ابری است که در آن بحث مدیریت حافظه یا Storage، مدیریت زیرساخت و بحث‌های networking با انتزاع بالایی به مصرف کاربر می‌رسد. به عبارت دیگر، تمامی مدیریت ای بخش‌ها بر عهده ارائه دهندگان است و ما اصلا با این بحث ها سروکاری نداریم. در واقع، هدف اصلی رایانش بدون سرور هم این است که این پیچیدگی‌ها را از کاربر بگیرد.

به طور کلی، یک پلتفرم بدون‌سرور را هر پلتفرم محاسباتی تعریف کرد که در آن مدیریت مستقیم سرور از کاربران مخفی شده و برنامه‌های کاربردی به صورت اتوماتیک در آن مقیاس‌پذیر می‌شوند و تنها هنگامی که در

[1] Serverless Computing
[2] abstract

حال استفاده از پلتفرم هستیم، هزینه آن را پرداخت می‌کنیم. [۱]

بسیاری از افراد، serverless و faas را معادل یک‌دیگر می‌دانند درحالی‌که اصلا این‌گونه نیست. در ادامه راجع به این بحث به طور مفصلی بحث خواهیم کرد اما باید بدانیم که این دو مقوله کاملا جدا از همدگیر هستند و مجددا تاکید می‌کنیم که رایانش بدون سرور یک مدل اجرایی در رایانش ابری است.

ازطرفی رایانش بدون سرور را باید نقطه مقابل رایانش سرور آگاهانه [۱] دانست که در آن از اطلاعات سرور در اختیار گرفته کاملا اگاهیم، کاملا بر مدیریت آن اشراف داریم و هرگونه تغییر از جمله متعادل‌سازی بارها، auto-scaling و ... باید توسط کاربر انجام شود.

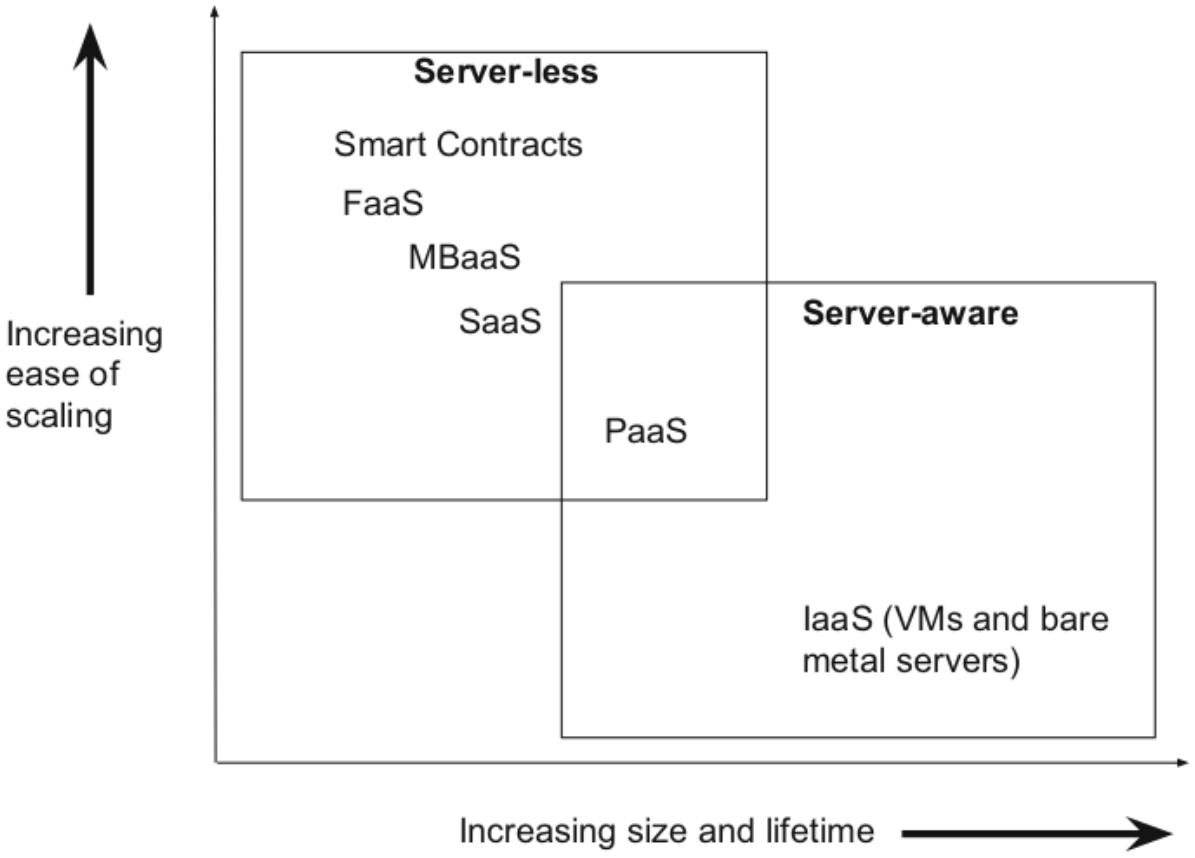
یک مثال از پیاده‌سازی رایانش سرور آگاهانه را در زیرساخت به عنوان سرویس [۲] یا به اختصار IaaS است. در نقطه مقابل در رایانش بدون سرور هیچ کنترلی بر روی سرور نداریم، تنها می‌توانیم یک برنامه را بر روی سرور اجرا کنیم یا اجرای آن را به حالت تعلیق درآورده یا آن را از روی سرور حذف کنیم که هیچ‌کدما از این موارد نیز به صورت مستقیم انجام نمی‌گیرد؛ بلکه رابط گرافیکی و API وجود دارد که از طریق آن‌ها این تغییرات را اعمال می‌کنیم. بنابراین در رایانش بدون سرور، عملا هیچ راهی برای مدیریت مستقیم سرور و زیرساخت نداریم.

شکل ۱.۲ مرزهای بین رایانش بدون‌سرور و رایانش سرور آگاهانه را نمایش می‌دهد.

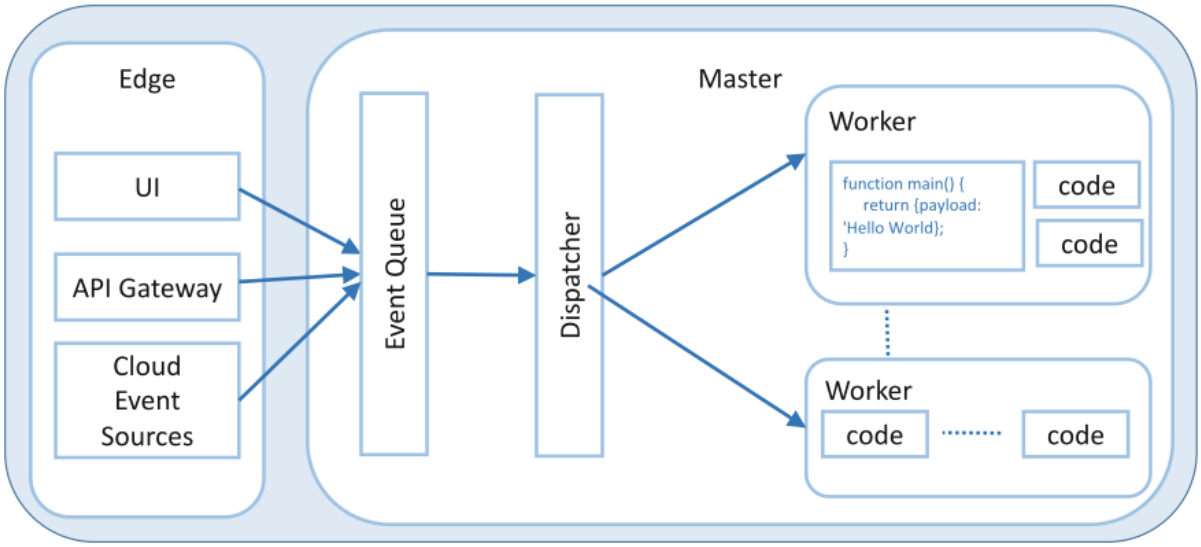
البته باید به این نکته توجه‌داشت که امروزه مرزهای بین رایانش سرور آگاهانه با رایانش بدون سرور در حال کمرنگ شدن و بعضا از بین رفتن است و این تقسیم بندی ابدا قاطعیت ندارد.همچنین تفکیک برخی موارد مانند Platform-as-a-Serice یا به اختصار PaaS به راحتی انجام نمی‌گیرد بلکه این نوع رایانش می‌تواند از نوع باسرور یا بدون سرور باشد. در این شکل هرچه به سمت محور افقی حرکت می‌کنیم دانه‌بندی و طول‌عمر افزایش پیدا می‌کند و هرچه به سمت بالاتر می‌رویم، scalingz راحت‌تر انجام می‌گیرد.

از مزایای رایانش بدون سرور همچنین می‌توان به پشتیبانی و توسعه راحت‌تر اپلیکیشن‌ها با معماری میکروسرویس و نانوسرویس هم اشاره کرد. البته معماری نانوسرویس مبحث جدیدتری است و جای پژوهش‌های بیشتری دارد.

[1] Server Aware
[2] Infrastructure as a service



شکل ۱.۲: مرزهای رایانش بدون سرور و رایانش سرورآگاهانه



شکل ۲.۲: معماری کلی یک پلتفرم بدون سرور

[۱]

۲.۱.۲ معماری

واژه serverless ممکن است این تفکر را به ذهن مبتدر سازد که اصلا در این نوع مدل رایانشی سروری نداریم؛ در حالی‌که این امر بسیار اشتباه است. در رایانش بدون سرور اگرچه سروری برای مدیریت به کاربر اختصاص داده نمی‌شود اما این موضوع بدان معنا نیست که اصلا سروری در کار نیست. در واقع مانند تمامی مدل‌های رایانشی در این‌جا هم سرور داریم، ولی تمامی تصمیمات مثل توزیع بار، تعداد اپ‌های روی سرور، انتخاب سرورها برای اجرای اپلیکیشن، کانفیگ CI/CD و ... بر عهده ارائه دهنده خدمات ابری است. اما برای درک نحوه کار یک پلتفرم بدون سرور، بهتر است با معماری آن آشنا باشیم. شکل ۲.۲ معماری یک پلتفرم خدمات ابری را نشان می‌دهد.

همانگونه که در تصویر مشخص است دوبخش کلی داریم، بخش لبه[۲] و بخش رییس [۳]، بخش لبه شامل رابط گرافیکی کاربر[۴]، Gateway API و Cloud Event Source می‌شود که برای تعامل با سرور رییس[۵] مناسب هستند. از طرف دیگر، در سرور رییس، درخواست‌ها ابتدا به صف رخدادها[۶] رسیده. صف رخدادها مسئول مدیریت رخداد

[2] edge
[3] master
[4] User Interface
[5] master server
[6] Event Queue

و نظم‌دهی به آن‌ها است. هر داده در صف رخ‌داد نوبت‌دهی می‌شود و سپس به بخش توزیع‌کننده[۱] می‌رود. توزیع‌کنند آپلیکیشن را برای دیپلوی، یا درخواست را برای سرویس‌دهی به یک نود کارگر[۲] هدایت می‌کند نود کارگر نیز با ارسال کدهای پاسخ[۳] با سرور رییس در ارتباط است. [۵]

البته بهتر است بدانیم که امروزه ادبیات رییس-کارگر[۴] برای نامیدن این معماری منسوخ شده و به جای آن از ادبیات رییس-گره[۵] استفاده می‌کنند.

۳.۱.۲ ویژگی‌های پلتفرم‌های بدون سرور

امروزه پلتفرم‌های بدون سرور بسیاری وجود دارند که روزانه بر تعداد آن‌ها افزوده می‌شود. اما باید دنبال ویژگی‌هایی برای آن‌ها باشیم که براساس آن‌ها بتوان این پلتفرم‌ها را تفکیک کرد و دست به مقایسه‌ی آن‌ها زد.

شاخص‌هایی که در این قسمت بررسی می‌کنیم شاخص‌های کمی و کیفی برای مقایسه‌ی بین این پلتفرم‌ها است.

۱. هزینه:

به طور معمول در رایانش بدون سرور، از مدل پرداخت پرداخت-به-ازای-استفاده[۶] استفاده می‌کنیم. یک ویژگی اساسی در تمایز بین ارائه‌دهندگان مختلف خدمات ابری، تفاوت آن‌ها در پشتیبانی از ویژگی scale-to-zero در این مدل محاسباتی[۷] است. این مورد باعث تفاوت معنی‌داری از هزینه‌ها در استفاده از پلتفرم‌های مختلف می‌شود. برخی از پلتفرم‌ها هم متن‌باز[۸] هستند که در این صورت، می‌توان به آسانی آن‌ها را بر روی ماشین مجازی یا سرور شخصی خود پیاده سازی کرد و برای استفاده از خدمات آن متحمل هیچ‌گونه هزینه‌ای نشد.

۲. کارایی و محدودیت‌ها[۹]

[1] dispatcher
[2] worker node
[3] Response Code
[4] Master-worker
[5] Master-Node
[6] pay-as-you-go
[7] Computational Model
[8] Open Source
[9] Performance and Limits

ارائه دهندگان مختلف، محدودیت‌های مختلفی را هم بر روی پلتفرم‌های مختلف خودشان اعمال می‌کنند. این محدودیت‌ها می‌توانند تعداد همزمان درخواست ها (number of concurrent)، حداکثر (requests استفاده از RAM و CPU د توسط یک تابع، حداکثر زمان زنده‌ماندن بعد از اجرا توسط تابع و ... است. البته برخی از محدودیت‌ها را می‌توان با صرف هزینه یا خرید پلن‌های درامدی سطح بالاتر برطرف کرد. مثلا پلتفرم functions Lambda AWS می‌تواند با صرف هزینه‌ی بیشتری حداکثر تعداد درخواست را افزایش داد. درحالی‌که این مورد در پلتفرم‌های اوپن سورس وجود ندارد. در حالت کلی پلتفرم‌های متن بازی مثل openfaas محدودیت‌های بیشتری را برای کاربر اعمال می‌کنند. علت این امر می‌تواند این باشد که این پلتفرمّ‌ها از نظر تکنولوژی و بازدهی (performance) کاملا از همتایان تجاری خود عقب هستند.

۳. زبان برنامه‌نویسی[1]

پلتفرم‌های بدون سرور از گستره‌ی عظیمی از زبان‌های برنامه‌نویسی پشتیبانی می‌کنند که شامل جاوااسکریپت، گو، پایتون، جاوا، سی‌شارپ، سویفت و پی‌اچ‌پی می‌شود. اکثر پلتفرم‌ها حداقل از ۵ زبان برنامه‌نویسی پشتیبانی می‌کنند. همچنین بسیاری از پلتفرم‌ها مستقل از زبان[2] هستند. یعنی درحالی‌که در داخل کانتینر اجرا می‌شوند (مثلا کانتینرهای داکر)، دیگر زبان برنامه‌نویسی برای آن‌ها اهمیتی ندارد. این پلتفرم‌ها توابع را در داخل کانتینر اجرا می‌کنند و نتیجه را برمی‌گردانند.

۴. مدل برنامه‌نویسی [3]

مدل‌های مختلفی برای تولید یک متد در پلتفرم‌های بدون سرور داریم. شیوه متداول استفاده از یک متد به نام main است که درون آن تابع اصلی تعریف می‌شود. همچنین معمولا ورودی‌های تابع در قالب شیئ‌های json تعریف می‌شوند.

۵. ترکیب توابع[4]

روش‌های گوناگونی برای اینکه یک تابع یا جریان کاری پیچیده‌را پیاده سازی کنیم وجود دارد. یک روش

[1] Programming Languages
[2] Language Independent
[3] Programming Model
[4] Compositions

استفاده از ترکیب‌های توابع است. تا به حال ۷ ترکیب مختلف شناسایی شده است. پلتفرم‌های تجاری [۱]هایی برای پیاده سازی این ترکیبات در خود تعبیه کرده‌آند. متاسفانه پلتفرم‌های متن باز مثل [۲] از این ترکیب پشتیبانی نمی‌کنند. در ادامه و در بخش کارهای مرتبط این ویژگی‌ها مطرح خواهند شد. کاربرد اصلی ترکیب توابع پیاده‌سازی عملکردهای پیچیده در پلتفرم بدون سرور است.

۶. استقرار[۳]

پلتفرم‌ها سعی می‌کنند پیاده‌سازی‌ها در رایانش بدون سرور را تا حد ممکن ساده کنند. این یکی از دلایل به وجود آمدن این مدل رایانشی بوده. به صورت معمول پلتفرم‌ها برنامه‌ها را در قالب کانتینرهای داکری دریافت می‌کنند و درون کانتینر مربوطه کد را اجرا می‌کنند. علاوه بر داکر پلن‌هایی از جمله دریافت کد باینری، دریافت سورس کد و سپس کانینرایز کردن آن وجود دارد.

۷. امنیت و حسابدری[۴]

این دو مورد در کنارهمدیگر به کار برده می‌شوند که معمولاخارج از بحث‌های رایانش ابری کاملا جدا از همدیگر به کار برده می‌شوند. در رایانش بدون سرور لازم است که اپ‌ها کاملا از همدگیر جدا اجرا شوند به این دلیل که بتوانیم برای هر کاربر هزینه‌ای که باید پرداخت کند را محاسبه کنیم. درصورتی که اجرای کاربران از همدگیر تفکیک شده نباشد، محاسبه‌ی هزینه ممکن نیست. اما اجرای جداگانه‌ی توابع از یکدیگر علت دیگری نیز دارد، امنیت. لازم است که توابع جداگانه اجرا شوند تا در توابع و کاربران نتوانند در کارهای همدیگر دخالتی داشته باشند. این مورد حتی می‌تواند باعث به وجود آمد باگ‌های امینتی و دسترسی کاربران به سیستم کاربران دیگر از طریق مدل رایانشی ما شود.

۸. پایش و اشکال زدایی[۵]

هر پلتفرم رایانشی امکاناتی از جمله پایش اولیه برای درخواست‌ها را به کاربر می‌دهد. البته این بحث یکی از مسائل باز در این حوزه است و نیاز به بررسی بیشتری دارد. در حال حاضر دیباگینگ از طریق تجزیه

[1] orchestrator
[2] openfaas
[3] Deployments
[4] Security and Accounting
[5] Monitoring and Debugging

و تحلیل لاگ‌های سیستم ممکن است ولی ممکن است در آینده بهبودهایی در این حوزه حاصل شود. علت اینکه دیباگینگ بسیار چالش برانگیز است این است که در رایانش بدون سرور اپ‌های ما کانتینرایز می‌شوند و چون محیط کانتیر محیطی ایزوله است، امکان مطالعه و دیباگینگ ممکن نیست. بعلاوه توابع تنها در حالت استفاده در پلتفرم زنده هستند؛ پس مدت زمان اشکال‌زدایی ما نیز بسیار محدود می‌شود. باید به این نکته دقت داشت که به طور متوسط در رایانش بدون سرور، توابع ۹۹درصد زمان را از تاریخ استقرار روی سرور، درخواب هستند. اما در مورد پایش نرم‌افزار پلتفرم بدون سرور در داشبورد مدیریتی خود امکاناتی جهت مشاهده منابع مصرف شده، منابع آزاد مدت زمان استفادهشده، تعداد درخواست‌ها و فراخوانی ها، تعداد شروع‌های سرد و ... دارد. به‌علاوه ابزارهای پایش مانند prometheus به خوبی با این پلتفرم‌ها امکان اتصال دارند و با پنل‌هایی مانند grafana می‌توان از مانیتورینگ مضاعف برای این سرورها بهره برد.

۴.۱.۲ پلتفرم‌های تجاری

پلتفرم‌های اندکی برای این قسمت وجود دارد. معروف‌ترین آن‌ها عبارتند از : AWS Lambda Funcitons ، Google Cloud Functions، Microsoft Azure Functions و IBM OpenWhisk

۱. AWS Lambda Functions

پلتفرم AWS [۶] پلتفرم ارائه شده در بحث رایانش بدون سرور بود که دارای خلاقیت‌های بسیاری بود. از مدل‌برنامه‌نویسی، مدل هزینه‌ای، محدودیت منابع، امنیت و مانیتورینگ مخصوص خود استفاده می‌کند. همچنین AWS از زبان‌هایJava، Node.js، Python و سی‌شارپ پشتیبانی می‌کند. این پلتفرم ارتباط خوبی با سایر خدمات و سرویس‌های AWS دارد و در این اکوسیستم اصطلاحا حل شده است.

۲. Functions Cloud Google

پلتفرم شرکت گوگل با نام Google Cloud Functions [۷]به تازگی از حالت آلفا خارج شده. این سرویس از زبان‌های بسیاری ساپورت نمی‌کند ولی به خوبی به درخواست‌های HTTP و HTTPS پاسخ می‌دهد. درحال حاضر اگرچه عملکرد محدودی برای این پلتفرم شاهد هستیم ولی با توجه به سابقه گوگل و معماری

متفاوت این پلتفرم، آینده خوبی برای آن می‌توان متصور بود. این پلتفرم هنوز به خوبی با سرویس‌های رایانش ابری گوگل ارتباط برقرار نکرده و جای کار بیشتری دارد.

۳. Functions Azure Microsoft

پلتفرم بعدی، پلتفرم Microsoft Azure Functions [۸]است. این پلتفرم وب‌هوک‌های HTTP را برای تعامل با کاربر پیاده سازی کرده است. از زبان‌های Bash، PHP، Python، Node.js، سی‌شارپ و اف‌شارپ یا هر زبان اجرایی (چون از کانتینترهای داکری استفاده می‌کند) پشتیبانی می‌کند. بخشی از کدها و پروژه‌های انجام شده با این پلتفرم توسط مایکروسافت در گیت‌هاب این پروژه متن‌باز شده اند. همچنین برای راحتی دیباگنگ مایکروسافت در CLI مربوطه امکان Caching یا استفاده از حافظه موقت را گنجانده است. این پلتفرم به مقبولیت قابل قبولی در بین جوامع توسعه‌دهندگان رسیده و روز به روز بر امکانات آن افزوده می‌گردد.

۴. OpenWhisk Apache

پلتفرم آخر، پلتفرم OpenWhisk [۹]است که در برابر پلتفرم‌های دیگر البته بسیار ساده‌تر به نظر می‌رسد. این پلتفرم اپن‌سورس توسط شرکت IBM تولید و پشتیبانی می‌شود. از قابلیت استفاده زنجیره‌ای توابع بهره‌می‌برد و در مبحث Orchestration توابع از پلتفرم‌های رقیب خود جلوتر است (منبع به مقاله ۱). همچنین OpenWhisk توانایی اجرای هر تابعی را دارد؛ زیرا از داکر به عنوان runtime نیز استفاده می‌کند. سورس این پروژه در آدرس گیت‌هاب OpenWhisk موجود است. در شکل زیر نیز می‌توان معماری آن را مشاهده کرد.

همانگونه که در شکل بالا مشخص است. این معماری خیلی به معماری مینیمال یک پلتفرم بدون سرور شبیه است. البته در مقایسه با شکل قبل امکانات بیشتری از جمله امنیت، مانیتوریگ و لاگ‌گیری را اضافه کرده است.

۱. Apache OpenWhisk

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد

نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

۲. Openfaas

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

۳. OpenLambda

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود

طراحی اساسا مورد استفاده قرار گیرد.

۵.۱.۲ پلتفرم‌های آزاد و متن باز

علاوه بر موارد فوق پلتفرم‌های متن بازی برای رایانش بدون سرور ارائه شده که در ادامه شرح خواهیم داد.

فصل ۳

کارهای مرتبط

لورم ایپسوم متن ساختگی با تولید سادگی نامفهوم از صنعت چاپ، و با استفاده از طراحان گرافیک است، چاپگرها و متون بلکه روزنامه و مجله در ستون و سطرآنچنان که لازم است، و برای شرایط فعلی تکنولوژی مورد نیاز، و کاربردهای متنوع با هدف بهبود ابزارهای کاربردی می باشد، کتابهای زیادی در شصت و سه درصد گذشته حال و آینده، شناخت فراوان جامعه و متخصصان را می طلبد، تا با نرم افزارها شناخت بیشتری را برای طراحان رایانه ای علی الخصوص طراحان خلاقی، و فرهنگ پیشرو در زبان فارسی ایجاد کرد، در این صورت می توان امید داشت که تمام و دشواری موجود در ارائه راهکارها، و شرایط سخت تایپ به پایان رسد و زمان مورد نیاز شامل حروفچینی دستاوردهای اصلی، و جوابگوی سوالات پیوسته اهل دنیای موجود طراحی اساسا مورد استفاده قرار گیرد.

فصل ۴

نتیجه‌گیری

در فصل قبل در ارتباط با راهکارهای ارائه شده توسط هر یک از مقاله‌ها به تفصیل، تشریح نمودیم. در این فصل می‌خواهیم به بیان نتایج و مقایسه راهکارهای ارائه شده بپردازیم. در ابتدا به مقایسه‌های روش‌های مبتنی بر مکانیزم DRX/DTX با یکدیگر می‌پردازیم.

مراجع

[1] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol.62, p.44-54, Nov. 2019.

[2] P.-M. Lin and A. Glikson, "Mitigating cold starts in serverless platforms: A pool-based approach," *arXiv preprint arXiv:1903.12221*, 2019.

[3] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov, "Agile cold starts for scalable serverless," in *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[4] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, pp.205-218, 2020.

[5] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, *et al.*, "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*, pp.1-20, Springer, 2017.

[6] A. AWS, "Aws lambda functions," `https://aws.amazon.com/lambda/`.

[7] G. INC., "Google cloud fucntions," `https://cloud.google.com/functions`.

[8] Microsoft, "Microsoft azure platform," `https://azure.microsoft.com/en-us/services/functions/`.

[9] A. openwhisk corporation, "Ibm openwhisk platform," `https://openwhisk.apache.org/`.