بسمه تعالی

طراحی و توسعه ابزارهای خودکارسازی تست و نظارت بر محصول هیولا

گزارش پیشرفت کار سوم (۱۰۰ ٪)

مجری طرح: مهندس محمد تقوا

استاد راهنما : دکتر زینب نخعی

ناظر فنی پروژه: مهندس ابوالفصل شفیعی

آبان ۱۴۰۳

فهرست مطالب:

فهرست اشکال:

چکیده

هیولا یک بستر نرم‌افزاری برای ذخیره‌سازی و مدیریت داده‌ها در محیط‌های ابری است. داده‌ها در قالب شیء ذخیره و مدیریت می‌شوند و هر شیء شامل داده، مشخصات و هویتی منحصر به فرد است. ذخیره‌سازی شیء در ابر باعث ارتقای مقیاس‌پذیری و دسترس‌پذیری و انعطاف بیشتر در برابر خرابی‌ها شده و امکان بازیابی ایمن اطلاعات را فراهم می‌کند. برای رسیدن به پیکربندی‌های مناسب درسطح برنامهوسیستم‌عامل نیاز به تحلیل و انجام حجم بالایی از انواع تست‌ها داریم. با انجام تست‌ها و تحلیل نتایج و رسیدن به پارامترهای تاثیرگذار، می‌توان کارایی هیولا را در محیط‌های عملیاتی به حداکثر رساند. در حال حاضر برای رفع این چالش ها از چندین نرم افزار غیر یکپارچه و بدون ساختار معماری مشخصی استفاده می‌شود. این کار موجب به وجود آمدن خطای انسانی، کاهش کارایی و کندی سرعت فرآیندهای تست و بررسی محصول هیولا می‌شود.

برای رفع نیازهای مرتبط با تست و نظارت بر وضعیت سامانه ذخیره سازی، مجموعه ابزاری برای نظارت و تست هیولا به نام کارا (Kara) طراحی و توسعه داده شده است. کارا توانایی تست سامانه هیولا و مستند سازی از وضعیت آن را در طول تست و رخدادها دارد. این مجموعه دارای شش ابزار عملیاتی و یک ابزار مدیریتی است. در این پروژه سعی بر این است که فرآیند تست و نظارت خودکارسازی شود تا موجب به حداقل رسیدن خطای انسانی، صرفه جویی در زمان، کاهش نیاز به نیروی انسانی و همچنین کاهش هزینه محصول هیولا شود.

کلمات کلیدی: هیولا، کارا، تست، سند، گزارش، سوئیفت

فصل ۱

مقدمه

۱-۱- بیان مسئله:

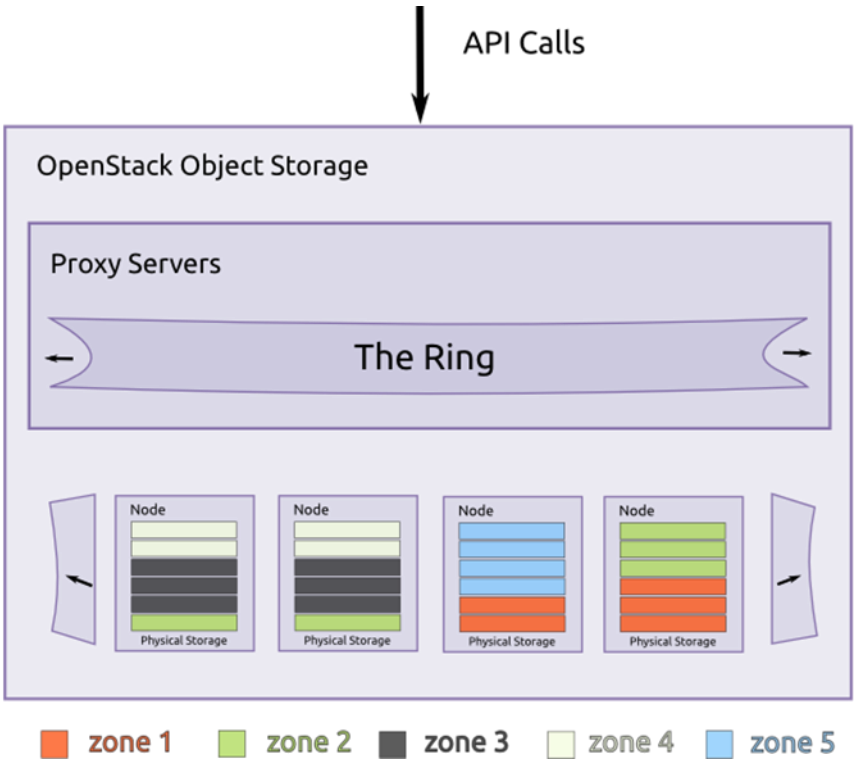
سامانه هیولا فضای ذخیره‌سازی مبتنی بر نرم افزار برای داده‌های غیرساخت‌یافته و به عنوان جایگزینی برای فضاهای ذخیره‌سازی مبتنی بر سخت افزار، پیچیده و گران قیمت است. شبکه ذخیره سازی توزیع‌شده و غیرمتمرکز هیولا امکان ذخیره سازی و پردازش همزمان حجم بالایی از اطلاعات را بدون ایجاد یک گلوگاه واحد فراهم می‌کند. استفاده از الگوی طراحی پراکسی بصورت توزیع‌شده و غیرمتمرکز امکان مقیاس‌پذیری خطی را در ذخیره و بازیابی اطلاعات بر اساس نیازمندی‌های مختلف از نظر سطوح دسترسی، نوع دسترسی، نوع پردازش و غیره فراهم می‌کند.

تمام نودهای پراکسی به یک تابع مشترک ثابت برای آدرس‌دهی فایل‌ها دسترسی دارند که قدرت تحمل خطای شبکه را از نظر تعداد نقاط دسترسی به میزان دلخواه قابل مدیریت می‌کند. در سامانه هیولا، سرویس‌های توزیع‌شده و غیرمتمرکز (شامل ۴ سرویس اصلی و چندین سرویس فرعی) برای مدیریت نرخ تکرار وجود دارند که بر روی هر یک از نودهای شبکه بصورت مستقل نصب می‌شوند. به این ترتیب، امکان دسترس‌پذیری بالا و پایداری داده‌ها در سطح شبکه بین نواحی دسترسی مختلف افزایش می‌یابد.

پلتفرم ابری هیولا خلق شده توسط شرکت برنا بر پایه سیستم ذخیره‌سازی اشیاء open stack swift ساخته شده است. ذخیره‌سازی شیء یک روش ذخیره‌سازی داده‌ها در محیط‌های ابری است که در آن داده‌ها در قالب شیء ذخیره و مدیریت می‌شوند. در این نوع از معماری اشیاء واحدهای گسسته‌ای هستند که در یک فضای صاف ذخیره‌سازی می‌شوند و بر خلاف معماری مبتنی بر فایل دیگر خبری از فولدرها، دایرکتوری ها و یا مسیرهای پیچیده نیست. هر شیء را می توان به عنوان یک ریپازیتوری ساده و در عین حال مستقل در نظر گرفت که شامل دیتا، متا دیتا (اطلاعات توصیفی مربوط به شیء) و یک شماره شناسایی منحصربفرد است. با این شماره شناسایی یکتا برنامه‌ها قادر خواهند بود تا به راحتی فایل‌ها و داده‌های مورد نیاز خود را یافته و به آن دسترسی پیدا کنند. در این مدل می‌توان با تجمیع شمار زیادی از دستگاه‌های ذخیره‌سازی شیء یک خوشه تشکیل داده و به راحتی این منابع را در سطح وسیع توزیع کرد. این کار این امکان را می‌دهد تا از مهم‌ترین ویژگی معماری ذخیره‌سازی اشیاء که همانا مقیاس پذیری نامحدود، انعطاف پذیری و بازیابی فاجعه[1] است نهایت بهره را برد.

دسترسی به اشیاء یا همان داده‌ها از طریق API (رابط برنامه نویسی برنامه) امکان پذیر است. API در نظر گرفته شده در سوئیفت یک restful API مبتنی بر http است و این امکان را می‌دهد تا به کمک query زدن بر اساس متا دیتای مربوط به شیء موردنظر از طریق اینترنت و از هر کجا و از روی هر دستگاهی به داده دسترسی داشته باشیم. از آنجایی که این API از دستورات http پشتیبانی می‌کند در نتیجه به راحتی می‌توان از طریق دستوراتی نظیر post برای به‌روزرسانی متادیتا، put برای ایجاد شیء، get برای بازیابی و دریافت و یا delete برای حذف یک شیء اقدام کرد. شکل ۱ معماری سطح بالای swift را نشان می‌دهد.

[1] disaster recovery

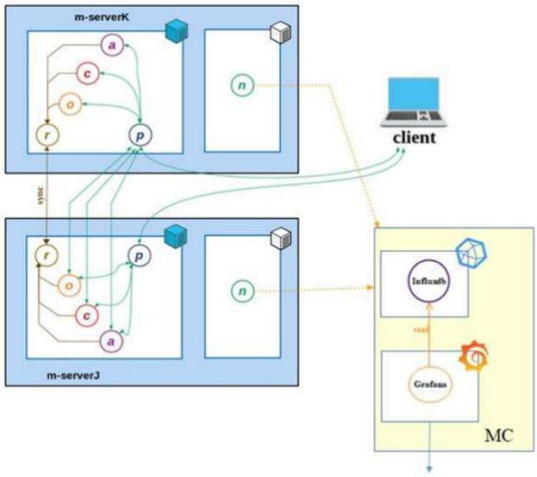


شکل ۱ : معماری *swift*

با توجه به ماهیت توزیع شده و پیچیده هیولا، به منظور نظارت بر عملکرد آن لازم است چندین سرویس برای ذخیره‌سازی متریک‌های جمع‌آوری شده از سامانه و نظارت بر سامانه راه‌اندازی شوند. در اینجا از ابزارهای نظارت Grafana و netdata و پایگاه داده ifluxdb استفاده می‌شود. این سه سرویس در کنار یکدیگر بخشی از وظایف سرور Master Controller(MC) را تشکیل می‌دهند که به عنوان سرویس هیولا ۳۶۰ در معماری این محصول (شکل ۲) شناخته می‌شود. وظیفه کلی این سرویس نظارت بر عملکرد هیولا است.

شکل ۲: معماری محصول

اطلاعات نظارتی هر نمونه هیولا ابتدا توسط statsd که یک سرویس نظارت ساده است و به آخرین تغییرات و اطلاعات سیستمی گوش می‌دهد جمع‌آوری می‌شود. سپس آنها را به سرویس netdata که یک نرم افزار پیشرفته‌تر نمایش و ذخیره اطلاعات و وضعیت سیستم است ارسال می‌کند. این اطلاعات دریافتی تا دو روز روی این سرویس ذخیره می‌شوند و برای نگه‌داری طولانی مدت به پایگاه داده influxdb ارسال می‌شود که یک نوع پایگاه داده سری زمانی است و در آن ذخیره می‌شود. در انتها برای نظارت دقیق سرویس هیولا این پایگاه داده به نرم افزار Grafana متصل می‌شود و می‌توان اطلاعات را در بازه‌های زمانی مختلف به صورت گرافیکی بررسی کرد و بسته به نیاز از آن یک خروجی برای تحلیل و مستندسازی دریافت کرد. در شکل ۳ ارسال اطلاعات به سرور نظارت نشان داده شده‌است.



شکل ۳: روند ارسال اطلاعات به سیستم نظارت هیولا

چالش های معماری فعلی نظارت و تست هیولا:

- در این فرآیند تست از ابزارهای بنچمارکینگ و نظارت مختلف استفاده می‌شود که هرکدام دارای فرآیند اجرا و پیاده سازی پیچیده و متفاوت از یک‌دیگر هستند و استفاده از این ابزارها نیازمند نیروی کار ماهر و زمان بسیار است.
- احتمال به وجود آمدن خطای انسانی در طی فرآیند تست و گزارش.
- دشوار بودن و زمانبر بودن دسته بندی و بایگانی کردن نتایج تست یا رخ داد های پیش آمده.
- وابستگی زیاد به ابزار های 3rd party و احتمال به وجود آمدن تداخلات نرم افزاری.
- افزایش هزینه و اتلاف وقت با توجه حجم نیروی کار انسانی مورد نیاز در این معماری.

با توجه به ساختار معماری پلتفرم ذخیره‌سازی ابری هیولا برای رسیدن به پیکربندی مناسب در سطح برنامه و سیستم عامل نیازمند تحلیل و انجام حجم بالایی از انواع تست‌ها هستیم.چون برای انجام تست‌ها نیاز داریم که معماری و پیکربندی‌های سرویس‌های هیولا، و پیکربندی‌های سیستم‌عامل میزبان را تغییر دهیم. تست‌های کارایی با اهداف مختلفی گرفته می‌شوند و بدون استفاده از

ابزارهای تست خودکار، تست کارایی موثر را نمی‌توان انجام داد. با این وجود به دلیل تعداد زیاد متغیرهایی که روی تست‌ها اثر می‌گذارد لازم است که این متغیرها دسته‌بندی شده و با ثابت در نظر گرفتن برخی از پارامترها، به بررسی تاثیر متغیرهای محدود پرداخته شود.

در حال حاضر فرآیند تست و بررسی به این صورت است که یک ابزار بنچمارکینگ تولید بارکاری را به شکل ارسال، دریافت و حذف اطلاعات روی هیولا با تعداد قابل تنظیمی از اشیاء شبیه سازی می‌کند. درست مانند زمانی که تعداد زیادی از کاربران به سرورهای پراکسی هیولا متصل هستند و این عملیات را روی داده‌های خود انجام می‌دهند. با توجه به سرویس‌های نظارتی موجود در پلتفرم ذخیره‌سازی که در بخش قبل توضیح داده شد، متریک‌های تعریف شده در نرم افزار netdata در زمان انجام تست در دیتابیس ذخیره و بعد از آن به شکل نمودارهای مورد نظر در سیستم نظارت نمایش داده می‌شود.

در نهایت برای تحلیل نتایج نیاز است که گزارش‌هایی نیز تهیه شود. لازم به ذکر است چنین اطلاعاتی در محیط‌عملیاتی نیز مورد استفاده قرار می‌گیرند و گزارش‌های دوره‌ای از وضعیت سلامت سامانه تهیه می شود. هر کدام از ابزارهای بنچمارک برای تولید بار کاری نیاز به پیکربندی دارند که اطلاعاتی از قبیل سایز و تعداد اشیاء و نوع درخواست را مشخص می‌کند. هر تست با توجه به هدفی که دنبال می‌کند بارکاری متفاوتی را نیاز دارد. بنابر این قبل از شروع تست این مشخصات به صورتی دستی بر روی ابزار تنظیم می‌شود که نیازمند شناخت کامل از ابزار و امکاناتی است که برای تولید بارکاری در اختیار قرار می‌دهد.

علاوه بر این ممکن است قبل از شروع هر تست نیاز به تغییر پیکربندی سرویس‌ها در سطح برنامه و حتی تغییر پیکربندی‌های سیستمی باشد. پس از انجام تست، با توجه به نیاز و هدف، خروجی‌ها به شکل جداول و نمودارها استخراج می‌شود تا مورد تحلیل‌های بعدی قرار بگیرد یا بایگانی شود. در حال حاضر، تمامی این مراحل به صورت دستی هستند و باید توسط یک یا چند نیروی انسانی انجام شود و باعث هدر رفت زمان و انرژی و هزینه بسیار می‌شود و ممکن است نتایج در معرض خطاهای انسانی قرار گیرد.

شرح نیازها:

- ایجاد انواع بارکاری برای اجرای تست های مختلف و شبیه سازی استفاده از محصول در محیط های عملیاتی.
- اجرای تست ها و بارکاری به صورت خودکار و ایجاد صفی از مجموعه آنها و دریافت نتایج تست و دسته بندی آنها.
- ذخیره نتایج تست و یا اتفاقات رخ داده در محیط های عملیاتی به صورت جداول csv و گراف تا تحلیل دقیق تری انجام شود و مستند سازی نتایج تحلیل ها.
- ذخیره سازی و دسته بندی اطلاعات تست و یا رخداد های محیط عملیاتی که شامل داده های دیتابیس در بازه های زمانی مشخصی نیز هستند و امکان بازیابی این اطلاعات و یا انتقال آنها نیز وجود داشته باشد.

۱-۲- راه حل پیشنهادی:

برای رفع چالش ها و نیازهای پیش رو مجموعه ابزاری برای نظارت و تست هیولا به نام کارا (Kara) طراحی و توسعه داده شده است که توانایی تست سامانه هیولا و مستند سازی از وضعیت آن را در طول تست و رخداد ها دارد. این مجموعه دارای شش ابزار عملیاتی و یک ابزار مدیریتی است. در این پروژه سعی بر این است که فرآیند تست و نظارت خودکارسازی شود تا در زمان و هزینه صرفه‌جویی شود و سرعت انجام تست‌ها و گزارش‌گیری در پلتفرم ابری هیولا افزایش یابد. این مجموعه ابزار کاربرد غیر تستی نیز دارند و در محیط‌های عملیاتی نیز قابل استفاده بوده و می‌توانند باعث تسریع در انجام کارها و کاهش خطای انسانی شوند. در مجموع تمام این ابزار ها رفع کننده نیاز های ذیل در پلتفرم هیولا هستند:

- Manager: این ابزار وظیفه مدیریت مجموعه ای از ابزار های موجود را دارد و آنها را بسته به نیاز کاربر در حالت های مختلف به صورت خودکار اجرا می‌کند
- Config_Gen: این ابزار وظیفه ساخت بارهای کاری مختلف یا کانفیگ‌های مختلف سرویس‌های هیولا را با استفاده از یک قالب دارد.
- Mrbench: این ابزار وظیفه تغییر کانفیگ های هیولا و سپس اجرای تست های کارایی هیولا را دارد. این تست ها با استفاده از ابزار cosbench اجرا می‌شوند.
- Status_Reporter: این ابزار وظیفه گزارش‌گیری از دیتابیس و نتایج تست و یا رخداد های موجود در هیولا را در قالب فایل های csv و عکس دارد.
- Monstaver: این ابزار وظیفه ایجاد نسخه پشتیبان و بازگردانی داده های موجود در بازه های زمانی تست و یا رخدادهای خاص از دیتابیس سامانه نظارتی هیولا را دارد.
- Analyzer: این ابزار وظیفه تحلیل و تجمیع گزارش های گرفته شده از هیولا و مقایسه و خطایابی در کانفیگ های آن را دارد.
- Report_recorder: این ابزار وظیفه مستندسازی نتایج تست ها و رخدادها یا گزارش‌های روتین را در سامانه مستندات دارد.

«کارا» توانایی انجام صفر تا صد یک فرآیند تست و ارزیابی و نظارت بر هیولا را دارد و می‌تواند این کار را به صورت کاملا خودکار و به دور از هرگونه خطای انسانی در کمترین زمان ممکن نسب به یک نیروی ماهر انجام دهد. معماری کارا به گونه ای طراحی شده است که هر یک از ابزار ها توانایی استفاده و کار کردن به صورت مجزا از یک دیگر را نیز داشته باشند و کمترین وابستگی بین آنها و دیگر نرم افزار های خارج از مجموعه کارا به وجود آید.

۱-۳- مفاهیم اولیه:

در این قسمت نرم افزار ها و کتابخانه هایی که در توسعه کارا مورد استفاده قرار گرفته اند توضیح داده شده اند.

Monster: یک راه حل ذخیره‌سازی تعریف شده توسط نرم‌افزار بر پایه شیء است که کوچکترین واحد ذخیره‌سازی داده در آن شیء نامیده می‌شود. نوع محتوا در هر شیء توسط کاربر مشخص می‌شود (به طور مثال: عکس، فیلم، اجرایی، متنی و ...) و هر شیء شامل مجموعه‌ای از فراداده است که جستجوی را امکان‌پذیر می‌سازد. این اطلاعات به طور دائم درون خوشه و معمولا روی دیسک سخت نگهداری می‌شوند.

هیولا دو مولفه منطقی اصلی دارد: سرویس پراکسی و سرویس ذخیره‌سازی. سرویس پراکسی مسئول ارتباطات بین مشتری و سرویس‌های ذخیره‌سازی و همچنین پیاده‌سازی بیشتر رابط‌های کاربری هیولا است. علاوه بر این، سرویس پراکسی مسئول مشخص کردن مکان فیزیکی داده در خوشه و ارسال پاسخ مناسب به مشتری است. سرویس ذخیره‌سازی مسئولیت ذخیره داده روی دیسک‌های ذخیره‌سازی، سرویس دادن به درخواست‌ها و رسیدگی به داده برای اطمینان از درستی آن را برعهده دارد. سرویس ذخیره‌سازی از یک طراحی روی دیسک به همراه معماری فهرست عمیق استفاده می‌کند. که به این دلیل برای خواندن یا نوشتن یک فایل نیاز به دسترسی مقدار تقریبا زیادی فراداده سیستم‌فایل وجود دارد.

Cosbench: یک ابزار تست استرس و بنچمارکینگ متن باز است که توسط شرکت اینتل برای آزمایش کارایی سیستم‌های ذخیره‌سازی اشیاء ابری توسعه یافته است. این نرم‌افزار به عنوان یک سیستم ذخیره‌سازی ابری سازگار با پروتکل swift می‌تواند برای انجام تست های معیار بر روی کارایی خواندن و نوشتن از هیولا استفاده شود. این نرم افزار از دو مولفه کلیدی تشکیل شده است:

- درایور(cosbench driver) : مسئول تولید بارکاری صدور عملیات برای هدف قراردادن ذخیره‌سازی اشیاء ابری و جمع آوری آمار کارایی.
- کنترلر(cosbench controller) : مسئول هماهنگی درایورها برای اجرای بارکاری، جمع‌آوری و یکپارچه سازی وضعیت زمان اجرا یا نتایج تست ها از نمونه‌های درایور و پذیرش ارسال های بارکاری.

Pywikibot: یک کتابخانه‌ی پایتون و مجموعه اسکریپت‌هایی است که خودکار روی سایت‌های مدیاویکی کار می‌کند. به‌طورکلی برای ویکی‌پدیا طراحی شده است. در حال حاضر از آن در پروژه‌های بنیاد ویکی‌مدیا و بسیاری از ویکی‌های دیگر استفاده می‌شود.

InfluxDB: یک دیتابیس از نوع Time-series و متن باز می‌باشد که توسط تیم InfluxData توسعه پیدا کرده است. این دیتابیس به کمک زبان Go توسعه پیدا کرده و شما می‌توانید بدون نصب هیچ‌گونه متعلقاتی آن را نصب کنید. برای جمع‌آوری داده‌ها محدودیتی پیش روی شما نیست و مهم نیس که داده‌های خود را چگونه و با چه فرمتی به آن می‌دهید. این پایگاه داده به طور خاص برای داده‌هایی که به صورت متوالی در طول زمان ثبت می‌شوند، مانند داده‌های سنسورها، لاگ‌ها، معلومات عملکرد سیستم‌ها، اطلاعات مانیتورینگ و دیگر داده‌های مشابه که نیاز به ثبت و نمایش تاریخچه تغییرات دارند، طراحی شده است.

Grafana: گرافانا یک پلتفرم قدرتمند برای ایجاد داشبوردها، تنظیم هشدارها، تولید گزارش‌ها و نظارت و تجزیه‌وتحلیل داده‌ها به‌صورت بلادرنگ (real-time) است. این پلتفرم از طیف گسترده‌ای از منابع داده پشتیبانی می‌کند، از جمله پایگاه‌داده‌های محبوب مانند MySQL،InfluxDB و همچنین سرویس‌های ابری مانند Monster و AWS . گرافانا در سال ۲۰۱۳ توسط تورکل ادگارد[۲] به عنوان یک ابزار برای تصویرسازی داده‌های سری زمانی ایجاد شد. طراحی این ابزار به منظور قابلیت سفارشی‌سازی و انعطاف‌پذیری بالا بوده و تمرکز آن بر ایجاد داشبوردهای خوش‌ظاهر و تعاملی است.

[2] Torkel Ödegaard

فصل ۲

مجموعه ابزار تست و نظارت (کارا)

در این فصل ابتدا به نصب و راه اندازی مجموعه ابزار کارا و سپس به شرح هر یک از ابزارها پرداخته می‌شود، این موارد شامل راهنمای استفاده از هر ابزار و مستندات توسعه آن می‌باشد.

۲-۱- روش های نصب و استقرار:

۲-۱-۱- docker :

در دایرکتوری kara دستورات زیر را اجرا کنید:

```
wget https://opengit.ir/smartlab/kara/-/blob/main/docker/docker-compose.yaml
-O docker-compose.yaml

docker-compose pull

docker-compose run --rm kara sudo /home/kara/docker/init.sh

docker-compose up -d
```

با اجرای این دستورات کانتینر کارا ایجاد و در حال اجرا قرار می‌گیرد. برای اطمینان از صحت اجرای دستورات بالا می‌توانید موارد زیر را دنبال کنید:

با اجرای دستور اول در دایرکتوری kara فایل docker-compose با محتوای زیر دانلود می‌شود:

```
# ~/kara/docker-compose.yaml
version: "3.8"

services:
  kara:
    image: registry.zdrive.ir/kara:1.0.1
    container_name: kara
    hostname: kara
    init: true
    restart: always
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - $PWD/backup:/tmp/influxdb-backup
      - $PWD/scenario_dir:/home/kara/manager/scenario_dir
      - $PWD/results:/home/kara/results
      - $PWD/kara-configs:/etc/kara
```

```
      - $PWD/jsons:/home/kara/status_reporter/jsons
      - $PWD/metrics:/home/kara/status_reporter/metrics
      - $PWD/rings:/home/kara/mrbench/rings
      - $PWD/workloads-configs:/home/kara/config_gen/workloads-configs
    ports:
      - "19088:19088"
      - "18088:18088"
    healthcheck:
      test: /home/kara/docker/healthcheck.sh | grep -w "OK" || exit 1
      interval: 10s
      timeout: 5s
      retries: 5

    networks:
      - karanet

networks:
  karanet:
    name: karanet
```

با اجرای دستور دوم ایمیج kara از رجیستری zdrive دریافت می‌شود که با دستور docker images میتوان از بارگذاری آن مخزن ایمیج‌ها اطمینان حاصل کرد:

```
docker images
REPOSITORY                      TAG             IMAGE ID       CREATED
SIZE
registry.zdrive.ir/kara         1.0.1           b96db567ad81   28 hours ago
1.28GB
```

در دستور سوم، پس از اجرای init.sh فایل های پیکربندی در دایرکتوری kara بصورت زیر ساخته می شوند:

```
$ ls -l kara/
total 36
drwxr-xr-x 2 root   root   4096 Nov 16 11:27 backup
-rw-rw-r-- 1 ubuntu ubuntu 1047 Nov 17 13:27 docker-compose.yaml
drwxr-xr-x 2 root   root   4096 Nov 16 11:27 jsons
drwxr-xr-x 2 root   root   4096 Nov 17 09:57 kara-configs
drwxr-xr-x 2 root   root   4096 Nov 16 11:27 metrics
drwxr-xr-x 6 root   root   4096 Nov 17 09:58 results
drwxr-xr-x 2 root   root   4096 Nov 17 13:28 rings
drwxr-xr-x 2 root   root   4096 Nov 17 09:58 scenario_dir
drwxr-xr-x 2 root   root   4096 Nov 17 13:28 workloads-configs
```

بسیاری از فایل های پیکربندی کارا به صورت volume تعریف شده اند. برای تغییر پیکربندی ها، می توانید خارج از کانتینر با دسترسی راحت تر به فایل ها تغییرات را اعمال کنید:

```
$ tree kara/
.
├── backup
├── docker-compose.yaml
├── jsons
│   ├── custom.json
│   ├── memory_panel.json
│   ├── network_panel.json
│   ├── Partial_Monitoring.json
│   └── Performance_Overview.json
├── kara-configs
│   ├── analyzer.conf
│   ├── monstaver.conf
│   ├── mrbench.conf
│   ├── report_recorder.conf
│   └── status_reporter.conf
├── metrics
│   ├── max_metric_list.txt
│   ├── mean_metric_list.txt
│   ├── min_metric_list.txt
│   └── sum_metric_list.txt
├── results
├── rings
├── scenario_dir
│   └── manager_scenario.yaml
└── workloads-configs

8 directories, 16 files
```

با اجرای دستور چهارم یک کانتینر از kara ایجاد می‌شود که برای سلامت سنجی کانتینر می‌توان از دستور docker ps استفاده کرد:

```
$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED
STATUS                  PORTS
NAMES
1653fd3ef043   kara:1.0.1          "/bin/sh -c 'cd $COS…"   3 hours ago   Up 3
hours (healthy)   0.0.0.0:18088->18088/tcp, :::18088->18088/tcp, 18089/tcp,
0.0.0.0:19088->19088/tcp, :::19088->19088/tcp, 19089/tcp
kara
```

با استفاده از دستورات زیر می‌توانید اقدام به حذف، متوقف یا راه‌اندازی مجدد کانتینر کنید.

```
docker-compose down
docker-compose stop
docker-compose restart
```

حال اقدامات پس از نصب را در قسمت فرآیند های پس از نصب در ادامه ی این فصل دنبال کنید.

۲-۱-۲- git:

۲-۱-۲-۱- نصب نرم افزار بنچمارکینگCosbench :

این برنامه یک ابزار تست استرس متن باز است که توسط شرکت اینتل برای آزمایش کارایی سیستم های ذخیره سازی اشیاء ابری توسعه یافته است.این نرم افزار به عنوان یک سیستم ذخیره سازی ابری سازگار با پروتکل swift می‌تواند برای انجام تست های معیار بر روی کارایی خواندن و نوشتن از هیولا استفاده شود.

برای آشنایی و راهنمای نصب می‌توانید به سند (Cloud_Object_Storage_Benchmark) مراجعه کنید.

پس از نصب به دایرکتوری اصلی می رویم و با استفاده از دستورات زیر، اسکریپت cli.sh را به حالت قابل اجرا در می آوریم و یک soft link در مسیر usr/bin/ ایجاد می کنیم.

```
sudo chmod +x /home/user/cosbench/0.4.2.c4/cli.sh
sudo ln -s /home/user/cosbench/0.4.2.c4/cli.sh /usr/bin/cosbench
```

۲-۲-۱-۲- کلون کردن آخرین ورژن از برنامه از opengit:

```
git clone https://opengit.ir/smartlab/kara
```

۲-۲-۱-۳- تنظیمات سیستم عامل: ساخت user مخصوص کارا در mc و تمام سرورهای هیولا و یا در موارد خاص استفاده از user های موجود با دسترسی sudo:

```
adduser kara
```

تغییر فایل sudoers و دادن دسترسی به کاربری که کارا اجرا می‌کند در سرور میزبان کارا و سرور های هیولا برای اجرای دستورات sudo بدون نیاز به password:

```
# visudo

    %sudo ALL=(ALL:ALL) ALL
    kara ALL=(ALL) NOPASSWD: ALL
```

۲-۲-۱-۴- اجرای ابزارconfigure : پس از انجام مراحل قبلی به دایرکتوری manager رفته و برنامه configure را اجرا کنید تا یکسری از فرآیندهای پیشنیاز اجرا و نصب کارا را انجام دهد.

```
bash configure.sh
```

۲-۲-۱-۴-۱- نصب کتابخانه های پیشنیاز ابزار:

نکته: فقط درصورتی که ابزار configure موفق به نصب آنها نشد اقدام به نصب آنها در سرور اجرا کننده کارا کنید.

```
apt install -y python pip sshpass
pip install pytz jdatetime datetime matplotlib pandas alive_progress
BeautifulSoup4 wikitextparser mwparserfromhell
```

۲-۲- فرآیند های پس از نصب:

۲-۲-۱- تغییرات در فایل MC-docker-compose :

در بعضی از گزارشات کارا از سرور های هیولا نیاز است تصاویری از گرافانا دریافت شوند برای این کار نصب این پلاگین ضروری است. برای نصب پلاگین image renderer گرافانا نیاز است این موارد ذکر شده در پایین به فایل اصلی docker-compose سرور mc افزوده شوند.

نکته: برای نصب ورژن پیشنهادی ۳.۵.۰ نیاز به داشتن گرافانا با ورژن بالاتر از ۷.۰.۰ است . گرافانا در ورژن های جدید خود ممکن است با ورژن های قدیمی influxdb به خوبی سازگار نباشد. برای رفع این مشکل بالاترین ورژن پیشنهادی گرافانا برای حداکثر سازگاری با پلاگین image renderer و influxdb و استفاده در کارا ورژن ۹.۳.۶ آن است.

```
image_renderer:
   image: grafana/grafana-image-renderer:3.5.0
   container_name: grafana-image-renderer
   hostname: renderer
   restart: always
   environment:
     - ENABLE_METRICS=true
     - HTTP_PORT=8081
     - RENDERER_LOG_LEVEL=debug
   ports:
     - 8081:8081
   networks:
     - mcnet
```

۲-۲-۲-تنظیمات سیستم عامل:

ابتدا وارد هرکدام از سرورهای هیولا و mc شده و کاربری که قرار است با استفاده از آن به سرور ssh زده شود و کارا با آن در ارتباط است را در فایل visudo همان سرور به صورت زیر اضافه کنید تا در هنگام ارتباط با سرور و اجرای دستورات نیازی به نوشتن رمز عبور درهنگام اجرا کارا نداشته باشد.

```
# visudo

    %sudo ALL=(ALL:ALL) ALL
    <your user> ALL=(ALL) NOPASSWD: ALL
```

۲-۲-۳- تغییر shard دیتابیس:

پس از نصب و راه اندازی اولیه کارا با یکی از دو روش ذکر شده به دایرکتوری /etc/kara/در سرور اجرا کننده کارا مراجعه کنید، سپس فایل کانفیگ monstaver.conf را ویرایش کرده و در بخش db_sources تنظیمات ssh و influxdb را مطابق تنظیمات فایل کانفیگ ابزار (monstaver -۲-۳-۵:) تغییر دهید تا shard دیتابیس ها به یک ساعت تغییر کند، این کار باعث افزایش راندمان و دقت در زمان بکاپ گرفتن و گزارش گیری کارا می‌شود.

حال به دایرکتوری ابزار manager رفته و آن را مشابه دستور ذیر اجرا کنید تا این عمل تغییر shard اجرا شود.

(فقط یک بار نیاز به انجام این کار برای همه دیتابیس ها است)

```
python3 manager.py -shard
```

نکته: پس از تغییرات shard در دیتابیس برای ذخیره سازی درست اطلاعات حداقل ۲ ساعت باید صبر کنید و پس از آن دیگر ابزار های کارا که مرتبط با دیتابیس هستند را اجرا کنید.

۲-۳- استفاده و توسعه ابزارها:

۲-۳-۱- ابزار مدیریتی manager:

این ابزار مانند مرکز کنترل سیستم نظارت عمل می‌کند و تمام ابزار های ساخته شده را به ترتیب پایپ لاین و سناریوهای مختلف اجرا می‌کند و ورودی‌های مورد نیاز هر ابزار را به آن می‌دهد. برای کاربری راحت‌تر و سریع‌تر، پیشنهاد می‌شود از ابزار manager استفاده شود تا دیگر ابزارها و نرم‌افزارها به واسطه‌ی این ابزار به صورت خودکار اجرا شوند و احتمال خطای انسانی کاهش یابد.

برای استفاده از این ابزار فقط کافی است یک فایل سناریو که با توجه به نیاز کاربر شخصی‌سازی شده است به ورودی manager داده شود.

فایل سناریو پیش فرض با فرمت yaml در دایرکتوری/manager/scenario_dir/ قرار دارد و می‌توان آن را بسته به نیاز ویرایش کرد و چندین نوع مختلف از آن ساخت. از ویژگی های مثبت استفاده از ابزار مدیریتی می‌توان به موارد ذیل اشاره کرد:

scenario_file.yaml

```
scenario:
        - Config_gen:
                {options}
        - Mrbench:
                {options}
        - Status-Reporter:
                {options}
        - Monstaver:
                {options}
        - Status_Analyzer:
                {options}
        - Report_Recorder:
                {options}
```

- مدیریت آسانتر ابزارها
- امکان ایجاد حالت های مختلف
- تعیین ورودی و خروجی هر ابزار
- مشخص کردن ترتیب اجرای ابزارها
- اجرای یک فرایند کامل به صورت خودکار
- مشخص کردن امکانات مورد نیاز در هر ابزار

شکل ۴: فایل سناریو

در فایل سناریو ترتیب اجرای ابزار ها مشخص شده که قابل تغییر است و در بخش هر ابزار ورودی‌ها و خروجی‌های مورد نیاز آن قابل تعریف است.

نکته: قبل از تنظیم سناریو و اجرای آن نیاز است فایل کانفیگ هر یک از ابزارها در مسیر /etc/kara/ تنظیم شود و اطلاعات مورد نیاز درون آنها قرارگیرد. برای این کار کافیست به بخش ذکر شده برای هر ابزار مراجعه کنید.

در زیر نمونه کانفیگ هر بخش از سناریو توضیح داده شده است و در بخش(۲-۳-۱-۱) نمونه هایی کاربردی از فایل سناریو آورده شده است.

- **Config_gen**:

- تنظیمات موجود برای ابزار در سناریو:

  - conf_templates: در این قسمت، لیستی از قالب های ورودی دریافت می‌شود. این لیست می‌تواند شامل چند قالب workload تست با شماره گذاری مشخص باشد، مانند نمونه زیر و یا شامل فایل های کانفیگ سرور های هیولا باشد.
  - output_path: مسیر خروجی برنامه output_path مشخص شده که برای هر قالب ورودی در آن یک دایرکتوری ساخته می‌شود که می‌تواند شامل چندین فایل باشد.

نکته: در صورتی که در مسیر خروجی ابزار فایل هایی از قبل وجود داشته باشد ابزار ابتدا سوالی در مورد حذف یا نگه‌داشتن فایل های قبلی می‌پرسد.

```
- Config_gen:
        conf_templates:    # list of cosbench and swift config file
            - /path/to/kara/config_gen/workloads.xml__1
            - /path/to/kara/config_gen/workloads.xml__2
            - /path/to/kara/config_gen/configs.conf
            - /path/to/kara/config_gen/object_swift.conf

        output_path: /path/to/kara/config_gen/out/
```

- **Mrbench**: این ابزار وظیفه اجرای تست ها را به ازای workload ها و کانفیگ های هیولا تولید شده را دارد.

- تنظیمات موجود برای ابزار در سناریو:

  - output_path: مسیر خروجی نهایی که درون آن یک دایرکتوری یکتا برای هر تست با نامی مشابه بازه زمانی آن تست ایجاد می‌شود دریافت شده است.
  - status_reporter: اگر بخواهیم این ابزار با اجرای هر تست به صورت جداگانه اجرا شود و خروجی تولید کند، باید نوع خروجی آن را(csv , img) مشخص کنیم، در غیر این صورت باید مقدار آن را none قرار دهیم.
  - Monstaver: اگر بخواهیم این ابزار با اجرای هر تست به صورت جداگانه اجرا شود و خروجی تولید کند، باید نوع خروجی آن را(backup , info) مشخص کنیم، در غیر این صورت باید مقدار آن را none

قرار دهیم. منظور از backup ، بکاپ دیتابیس و info گرفتن اطلاعات سخت افزاری و نرم افزاری حین تست است.

- conf_dir: مسیر قرار گرفتن خروجی های تولید شده توسط ابزار قبلی (config_gen) را ورودی می گیرد. در صورتی که قبل از این ابزار، ابزار config-gen در فایل سناریو باشد و اجرا شده باشد نیازی به استفاده از قسمت conf_dir نیست.
- ring_dirs: اگر بخواهیم رینگ های هیولا را تغییر دهیم و تست ها را با آن ها تکرار کنیم در این قسمت مسیر رینگ ها را قرار می دهیم.

نکته اول:

- **در صورتی که در مسیر خروجی ابزار فایل هایی از قبل وجود داشته باشد ابزار ابتدا سوالی در مورد حذف یا نگه‌داشتن فایل های قبلی می‌پرسد.**
- **اگر در هنگام اجرای این بخش از سناریو در ابزار mrbench به دلایلی فرآیند اجرای آن متوقف شود برنامه آخرین مرحله اجرا شده را ذخیره می‌کند و در اجرای بعدی از کاربر برای ادامه مراحل قبل یا شروع از ابتدا سوال می‌نماید.**

نکته دوم:

- **در زمان اجرای این ابزار در سناریو در صورت اجرای ابزار status_reporter و ساخت csv دو فایل تجمیع شده از تمام تست ها با نام های (merged , merged_info) نیز در مسیر خروجی و دایرکتوری analyzed ایجاد می‌شوند.**
- **فایل merged شامل تمام اطلاعات موجود در فایل های csv هر تست به علاوه اطلاعات workload و کانفیگ های swift در زمان تست است و فایل merged_info فقط شامل اطلاعات workload و کانفیگ swift در زمان تست است که این فایل فقط در زمان ثبت اسناد تست در کاتب استفاده می‌شود.**

```
    - Mrbench:
            output_path: /path/to/kara/results/

            # call status_reporetr and monstaver for each test
            status_reporter: csv,img    # values = none - csv - csv,img
            monstaver: backup,info    # values = none , backup,info - backup
- info

            #conf_dir: /path/to/kara/config_gen/out/
            ring_dirs:    # list of directory include ring files
                - /path/to/kara/mrbench/ring/r1/
                - /path/to/kara/mrbench/ring/r2/
                - /path/to/kara/mrbench/ring/r3/
```

- **Status_reporter**: اگر نیاز باشد این ابزار به صورت مستقل کار کند باید از قسمت مخصوص status_reporter در فایل سناریو استفاده کرد. این ابزار وظیفه گزارش‌گیری از دیتابیس و نتایج تست و یا رخداد های موجود در هیولا در قالب فایل های csv و عکس را دارد.

- تنظیمات موجود برای ابزار در سناریو:
  - time_list: برای ورودی بازه های زمانی چند روش وجود دارد. می توان لیستی از زمان های مورد نیاز را در فرمت های گوناگون نوشتاری (tehran timestamp) ورودی داد و یا فایلی که شامل لیستی از بازه های تست مورد نیاز است را قرار داد. (در تکه کد زیر قابل مشاهده است).
  - image: تصویر گراف از بازه های زمانی ذکر شده ساخته شود و یا خیر.
  - analyze_csv: تحلیل و تجمیع گزارش ها نیاز است یا خیر.
  - report_recorder: برای ثبت اطلاعات گزارش ها در کاتب است. برای گزارشات روزانه کاربرد دارد و در صورتی که به آن نیاز نباشد کل این بخش باید کامنت شود.
    - output_htmls_path: مسیر ذخیره فایل های html ساخته شده برای گزارش ها.
    - cluster_name: اسم کلاستر که در خروجی تاریخ گزارش به آن توسط ابزار اضافه می‌شود.
    - kateb_tags: رده های کاتب
    - kateb_list_page: اسم صفحه ای در کاتب که لیستی از نام سند های گزارش به آن اضافه می‌شود، این صفحه اگر وجود نداشت ساخته می‌شود اگر وجود داشت به انتهای آن لیست صفحات اضافه می‌شود.
  - output_path: مسیر خروجی نهایی گزارش های سرور های هیولا.

نکته: در این صورت استفاده از گزینه analyze_csv یک نمونه csv دارای تحلیل نیز از گزارش ها ساخت می‌شود پس حتما قبل از اجرای آنها ابزار status_analyzer را بسته به متریک ها یا measurement های موجود در فایل های متریک status_reporter حتما کانفیگ کنید.

```
- Status-Reporter:
    time_list:
        - now-1d,now
        #- now-24h,now
        #- "2023-09-10 23:59:59,2023-09-11 23:59:59"
        #- ./time.txt
    image: True
    analyze_csv: True
    report_recorder:
        output_htmls_path: path/to/kara/report_recorder/output_htmls/
        cluster_name: "کارا"
        kateb_tags:
            - "تست"
            - "کارایی"
```

```
                - "گزارشها"
            kateb_list_page: "name of a page"  # append all pages title
to this kateb page
            output_path: ./../results/
```

- **Monstaver** :

اگر نیاز باشد این ابزار به صورت مستقل کار کند باید از قسمت مخصوص monstaver استفاده کرد.

این ابزار وظیفه بکاپ گیری و بازیابی دیتابیس influxdb را دارد. همچنین می تواند تنظیمات مربوط به هیولا و اطلاعات سخت افزاری و نرم افزاری سرور ها را نیز ذخیره کند.

- تنظیمات موجود برای ابزار در سناریو:
  - input_path: لیستی از مسیر های دلخواه که نیاز هستند در فایل نهایی وجود داشته باشند مانند مسیر خروجی تست و گزارش.
  - time_list: برای ورودی بازه های زمانی بکاپ گیری چند روش وجود دارد. می توان لیستی از زمان های مورد نیاز را در فرمت های گوناگون نوشتاری (tehran timestamp) ورودی داد و یا فایلی که شامل لیستی از بازه های تست مورد نیاز است را قرار داد. (در تکه کد زیر قابل مشاهده است).
  - batch_mode: با فعال کردن حالت batch_mode پس از اجرای تمام تست ها در ابزار mrbench عملیات بکاپگیری برای یک دسته تست انجام شود.
  - operation: می‌توان نوع عملیات را مشخص کرد که بکاپ از دیتابیس باشد یا به همراه اطلاعات سخت افزاری و نرم افزاری سرور های هیولا و یا عملیات بازیابی.

نکته: در صورتی که قبل از monstaver از status_reporter استفاده شده باشد و بازه های زمانی آنها یکسان باشند می‌توان فقط یک لیست بازه زمانی برای status_reporter ایجاد کرد و ابزار monstaver نیز از همان استفاده کند برای جلوگیری از تکرار آنها در فایل سناریو.

```
    - Monstaver:
            input_path: ./../results
            time_list:
                #- now-1h,now
                #- now-3d,now-2d
                #- "2023-09-11 00:00:00,2023-09-11 23:59:59"
                #- ./time.txt
            batch_mode: True # value = True for all backup modes and False
for restore
            operation: backup # values = restore , backup,info - backup -
info
```

- **Status_analyzer** :

در بخش این ابزار می‌توان دو عملیات های تجمیع (merge) و تحلیل (analyze) فایل های csv مورد نیاز را مشخص کرد.

- تنظیمات موجود برای ابزار در سناریو:
  - output_path: مسیر خروجی نهایی فایل های تحلیل و تجمیع شده.
  - merge: انجام عملیات تجمیع.
  - merged_csv: برای تجمیع کردن فایل های csv گزارش های گرفته شده از دیتابیس (influxdb) سرویس نظارت هیولا یا هر نوع csv دیگری، می‌توان همه فایل ها را در یک دایرکتوری مشترک انتقال داد و یا لیستی از آنها و مسیرشان نوشت تا فایل تجمیع شده آنها در خروجی ذکر شده ذخیره شود.
  - analyze: انجام عملیات تحلیل.
  - keep_source_columns: نگهداشتن ستون های اولیه در فایل نهایی تحلیل و یا حذف آنها.
  - analyze_csv: برای بخش تحلیل نیز می‌توان همان فایل تجمیع شده یا هر نوع csv دیگری را برای ورودی آن انتخاب کرد و مشخص کرد.

```
    - Status_Analyzer:
            output_path: /path/to/kara/results/analyzed/
            merge: True
            merge_csv: "./../result/*"  # list of csv file -->
'/path/csv1,/path/csv3,/path/csv3,' or /path/*

            analyze: True
            keep_source_columns: False    # keep original columns in source
csv file
            analyze_csv: "/path/to/kara/results/analyzed/merged.csv"
```

- **Report_recorder** :

این ابزار وظیفه ایجاد اسناد در قالب html و آپلود آنها در کاتب را دارد.

- تنظیمات موجود برای ابزار در سناریو
  - create_html: عملیات ایجاد کردن html های جدید.
  - hardware_template: فایل html برای استفاده به عنوان قالب گزارش‌های سخت افزاری.
  - software_template: فایل html برای استفاده به عنوان قالب گزارش‌های نرم افزاری.

- monster_test: تنظیمات مربوط به گزارش تست.
  - report: فعال کردن این بخش و ایجاد فایل html گزارش تست.
  - merged: فایل تجمیع شده اطلاعات کامل همه تست های یک دسته.
  - merged_info: فایل تکمیلی تجمیع شده تست ها که فقط شامل اطلاعات workload و کانفیگ swift در زمان تست است.
  - images_path: استفاده از دایرکتوری والد همه نتایج تست برای استخراج تصاویر و درج آنها در سند.
- output_path: مسیر خروجی نهایی فایل های html ساخته شده.
- configs_dir: دایرکتوری بکاپ های گرفته شده برای استخراج اطلاعات نرم افزاری و سخت افزاری.
- upload_to_kateb: عملیات آپلود اطلاعات در کاتب.
- cluster_name: اسم کلاستر هیولا برای تیتر سند در کاتب.
- scenario_name: اسم سناریو اجرا شده برای تیتر سند در کاتب.
- kateb_list_page: نام یک صفحه موجود در کاتب را که نیاز داریم اسم صفحات ساخته شده در آن به صورت فهرست لیست شود.

نکته: در صورتی که این ابزار در یک سناریو همراه با monstaver یا mrbench اجرا شود دیگر نیازی به مشخص کردن آدرس فایل بکاپ در قسمت configs_dir نیست چون مسیر آن از ابزار قبلی دریافت می‌شود و همچنین اگر همراه با status_reporter و mrbench اجرا شود مسیر عکس های ساخته شده نیز خودکار از آن ابزار دریافت می‌شود و نیاز به استفاده از images_path در بخش این ابزار و در فایل سناریو نیست.

```
    - Report_Recorder:
            create_html: True

            hardware_template:
/path/to/kara/report_recorder/input_templates/hardware.html
            software_template:
/path/to/kara/report_recorder/input_templates/software.html

            monster_test:
                report: True
                merged: /path/to/kara/manager/merged.csv
                merged_info: /path/to/kara/manager/merged_info.csv
                images_path: /path/to/kara/results/

            output_path: /path/to/kara/report_recorder/output_htmls/
            configs_dir: /tmp/influxdb-backup/backup_dir/

            upload_to_kateb: True
            cluster_name: kara
            scenario_name: performance
```

```
            kateb_list_page: "name of page"  # append all pages title to this
kateb page
```

۲-۳-۱-۱- مثال هایی از فایل سناریو:

۱- سناریو کامل: اجرای ابزار ها از ایجاد فایل های کانفیگ swift و workload در cosbench تا اجرای تست و دریافت گزارش csv و تصویری گراف و بکاپ گیری از دیتابیس (influxdb) و اطلاعات سخت افزاری و نرم افزاری سرور های هیولا و تا تحلیل و تجمیع و ساخت سند در کاتب.

```
scenario:

    - Config_gen:
            conf_templates:     # list of cosbench and swift config file
                - /path/to/kara/config_gen/workloads.xml__1
                - /path/to/kara/config_gen/workloads.xml__2
                #- /path/to/kara/config_gen/configs.conf
                - /path/to/kara/config_gen/object_swift.conf

            output_path: /path/to/kara/config_gen/out/

    - Mrbench:
            output_path: /path/to/kara/results/

            # call status_reporetr and monstaver for each test
            status_reporter: csv,img     # values = none - csv - csv,img
            monstaver: backup,info     # values = none , backup,info - backup
- info

            #conf_dir: /path/to/kara/config_gen/out/
            ring_dirs:     # list of directory include ring files
                - /path/to/kara/mrbench/ring/r1/
                #- /path/to/kara/mrbench/ring/r2/
                #- /path/to/kara/mrbench/ring/r3/

    - Status_Analyzer:
            #output_path: /path/to/kara/results/analyzed/
            #merge: True
            #merge_csv: "./../result/*"  # list of csv file -->
'/path/csv1,/path/csv3,/path/csv3,' or /path/*

            analyze: True
            keep_source_columns: True     # keep original columns in source
csv file
            analyze_csv: "/path/to/kara/results/analyzed/merged.csv"

    - Report_Recorder:
            create_html: True

            hardware_template:
/path/to/kara/report_recorder/input_templates/hardware.html
```

```
            software_template:
/path/to/kara/report_recorder/input_templates/software.html

            monster_test:
                report: True
                merged: /path/to/kara/manager/merged.csv
                merged_info: /path/to/kara/manager/merged_info.csv
                images_path: /path/to/kara/results/

            output_path: /path/to/kara/report_recorder/output_htmls/
            #configs_dir: /tmp/influxdb-backup/backup_dir/

            upload_to_kateb: True
            cluster_name: kara
            scenario_name: complete_scenario
            kateb_list_page: "name of page"  # append all pages title to this
kateb page

log:
   level: info  # values = debug - info - warning - error - critical
```

۲- سناریو دریافت گزارش روزانه از هیولا:

```
        - Status-Reporter:
            time_list:
                - now-1d,now
                #- now-24h,now
                #- "2023-09-10 23:59:59,2023-09-11 23:59:59"
                #- ./time.txt
            image: True
            analyze_csv: True
            report_recorder:
                output_htmls_path: path/to/kara/report_recorder/output_htmls/
                cluster_name: "کارا"
                kateb_tags:
                    - "تست"
                    - "کارایی"
                    - "گزارش‌ها"
                kateb_list_page: "name of a page"   # append all pages title
to this kateb page
            output_path: ./../results/
```

۳- سناریو اجرای تست کارایی هیولا: در صورت ساخت فایل های تمپلیت cosbench و کانفیگ swift در زمان اجرای تست:

```
scenario:

    - Config_gen:
            conf_templates:    # list of cosbench and swift config file
                - /path/to/kara/config_gen/workloads.xml__1
                - /path/to/kara/config_gen/workloads.xml__2
```

```
                #- /path/to/kara/config_gen/configs.conf
                - /path/to/kara/config_gen/object_swift.conf

            output_path: /path/to/kara/config_gen/out/

    - Mrbench:
            output_path: /path/to/kara/results/

            # call status_reporetr and monstaver for each test
            Status_Reporter: csv,img     # values = none - csv - csv,img
            monstaver: backup,info     # values = none , backup,info - backup
- info

            #conf_dir: /path/to/kara/config_gen/out/
            ring_dirs:     # list of directory include ring files
                - /path/to/kara/mrbench/ring/r1/
                - /path/to/kara/mrbench/ring/r2/
                - /path/to/kara/mrbench/ring/r3/
```

در صورت استفاده از فایل های از قبل ساخته شده تمپلیت cosbench و کانفیگ swift در اجرای تست:

```
 scenario:

    - Mrbench:
            output_path: /path/to/kara/results/

            # call status_reporetr and monstaver for each test
            status_reporter: csv,img     # values = none - csv - csv,img
            monstaver: backup,info     # values = none , backup,info - backup
- info

            conf_dir: /path/to/kara/config_gen/out/
            ring_dirs:     # list of directory include ring files
                - /path/to/kara/mrbench/ring/r1/
                - /path/to/kara/mrbench/ring/r2/
                - /path/to/kara/mrbench/ring/r3/
```

۴- سناریو ساخت سند در کاتب: در صورت وجود داشتن فایل بکاپ از اطلاعات سخت افزاری و نرم افزاری هیولا و فایل های تجمیع شده از گزارش های تست و رخداد:

```
 scenario:

    - Report_Recorder:
            create_html: True

            hardware_template:
/path/to/kara/report_recorder/input_templates/hardware.html
            software_template:
/path/to/kara/report_recorder/input_templates/software.html

            monster_test:
```

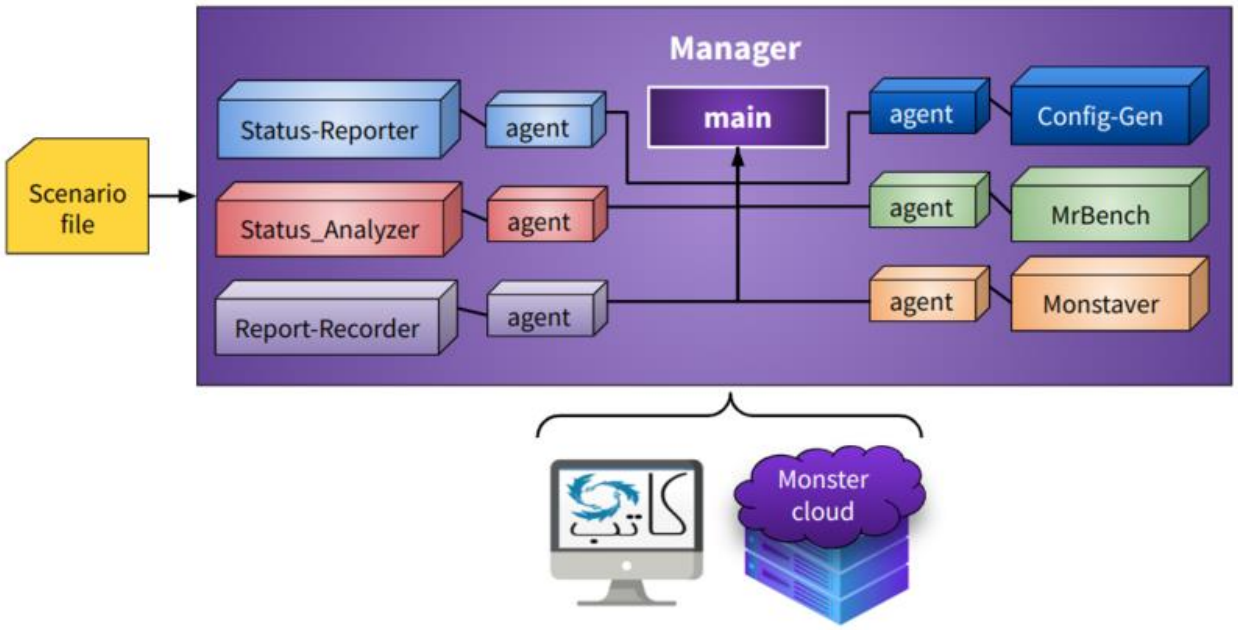
```
                report: True
                merged: /path/to/kara/manager/merged.csv
                merged_info: /path/to/kara/manager/merged_info.csv
                images_path: /path/to/kara/results/

            output_path: /path/to/kara/report_recorder/output_htmls/
            configs_dir: /tmp/influxdb-backup/backup_dir/

            upload_to_kateb: True
            cluster_name: kara
            scenario_name: performance
            kateb_list_page: "name of page"  # append all pages title to this
kateb page
```

۲-۳-۱-۲- روش استفاده از ابزار:

```
python3 manager.py -sn scenario_dir/manager_scenario.yaml
```

۲-۳-۱-۳- توسعه ابزار manager:

این برنامه به ازای هر یک از ابزار های کارا یک agent واسط دارد که وظیفه آماده کردن و دادن ورودی به ابزارها و گرفتن خروجی از آنها را دارد. برای اینکه بتوان از هریک از ابزارها در این برنامه استفاده کرد تمامی آنها به صورت ماژول های پایتونیimport شده اند .



شکل ۵:معماری manager

۲-۳-۱-۳-۱ شرح توابع موجود:

۱ـ load_config: این تابع فایل سناریو ورودی در فرمت yaml را در یافت و آن را با استفاده از کتابخانه yaml می‌خواند. متغییر data_loaded خروجی این تابع بوده و شامل کل فایل سنایو با همان ساختار اولیه است.

۲ـ config_gen_agent: این تابع واسط ابزار config_gen بوده بخش های مورد نیاز را از فایل سناریو قسمت config_gen را با استفاده از data_loaded برداشت می‌کند.

- بخش اول تابع : بررسی می‌شود که مسیر خروجی دارای فایلی از قبل هست یا خیر در صورت وجود داشتن از کاربر سوالی برای نگهداشتن یا حذف آنها از کاربر می‌پرسد در صورت حذف فایل های جدید جایگذین آنها می‌شود و در غیر این صورت خروجی های جدید در همانجا ذخیره و در صورت داشتن تشابه اسمی overwrite می‌شوند. در صورتی که کاربر پس از ۳۰ ثانیه به سوال پاسخ ندهد فایل های فعلی به یک دایرکتوری عقبتر در یک دایرکتوری (نام دایرکتوری اصلی + زمان اجرای برنامه) منتقل شده و فایل های جدید به جای آنها در میسر خروجی ذخیره می‌شوند.

```
def config_gen_agent(config_params):
        while True:

                #check output dir is empty or not and ask question for
keeping file

                if response == 'yes':

                        # remove current files

                elif response == 'no':

                        # Leaving existing contents untouched
```

- بخش دوم تابع : در این قسمت لیست فایل های تمپلیت ورودی در حلقه ای خوانده می‌شود و یک دایرکتوری جدید به ازای هر فایل ورودی با همان نام ساخته می‌شود و که خروجی های مختص هر فایل در به صورت مجزا در آن قرار گیرد سپس تابع main ابزار config_gen فراخوانی شده و ورودی های مورد نظر به آن داده می‌شود.

```
def config_gen_agent(config_params):
        for input_file in input_files:
                        firstConfNumber = 1

                # Create output directory for each input file

                workloads_configs = os.path.join(config_output,
os.path.basename(input_file).split('__')[0])
                    firstConfNumber = len(os.listdir(workloads_configs))+1

                config_gen.main(input_file_path=input_file,
output_directory=workloads_configs, conf_num=firstConfNumber)

return config_output
```

خروجی نهایی تابع مسیر دایرکتوری خروجی است که حالا شامل دایرکتوری و فایل های زیر مجموعه هر تمپلبت ورودی است.

۳ – تابع mrbench_agent: این تابع واسط ابزار mrbench است و ۳ ورودی دارد که شامل بخش مختص mrbench در فایل سناریو و مسیر خروجی نهایی تابع config_gen_agent و خود فایل سناریو است.

- بخش اول تابع: در این قسمت ابتدا یک hash از فایل فعلی سناریو ساخته می‌شود و به همراه چند نشانگر در فایلی ذخیره می‌شود که با هر بار اجرای برنامه این فایل بررسی شده و در صورتی که فایل فعلی سناریو با آخرین اجرا تفاوتی نکرده باشد و اجرای قبلی کامل خاتمه نیافته بود از کاربر سوال می‌کند که آیا نیاز به ادامه از آخرین سطح اجرای برنامه دارد یا خیر و پس از آن مسیر خروجی نهایی بررسی می‌شود که دارای فایلی از قبل هست یا نه و از کاربر سوال می‌شود نیازی به حذف فایل های فعلی هست یا خیر.

```
def mrbench_agent(config_params, config_file, config_output):
while True:
        #check output dir is empty or not and ask question for keeping
file

        if continue_response == 'yes':
           # Load previous state

        elif continue_response = 'no'
               # start mrbench from beginning

               while True:
                      #Do you want to continue last mrbench's test in
scenario?
               if response == 'yes':
                      #remove current files

               elif response == 'no':
                      # Leaving existing contents untouched
```

- بخش دوم تابع: در این قسمت سه حلقه تو در تو به ازای ring و swift_conf و workload وجود دارد که به ترتیب ذکر شده اجرا می‌شوند و طی این فرآیند دو دیکشنری conf_dict و swift_configs ساخته می‌شوند و پس از قرار دادن مقادیر مورد نیاز که شامل فایل های swift و ring و آدرس آنها هست هر دو دیکشنری تجمیع شده و به تابع copy_swift_conf در mrbench ارسال می‌شوند پس از آن فایل های workload به تابع submit در mrbench ارسال و خروجی آن یه صورت یک آدرس دیکشنری اطلاعات تست دریافت می‌شود.

```
def mrbench_agent(config_params, config_file, config_output):
for ri in range(R, total_ring_index):
     swift_rings[filename] = file_path
     for i in range(S ,Total_index):
            swift_configs[key] =
os.path.join(config_output,key,list_dir[(i//m)%len(list_dir)])
            if conf_exist or ring_exist:
                   merged_conf_ring = {**swift_rings, **swift_configs}
                   ring_dict = mrbench.copy_swift_conf(merged_conf_ring)
            for wi, test_config in
enumerate(sorted(os.listdir(conf_dict["workloads.xml"]))[W:], start=W):
                   cosbench_data, result_path = mrbench.submit(test_config_path,
result_dir)
```

- بخش سوم تابع: در این قسمت یک فایل yaml برای اطلاعات workload و swift و یک فایل yaml برای اطلاعات ring در فرمت هایی مشخص ساخته می‌شود و از تمام این ۳ نوع اطلاعات یک دیکشنری ایجاد می‌شود برای ارسال به تابع merge_csv در ابزار analyzer البته قبل از ارسال آن ابزار status-reporter در فایل سناریو و بخش mrbench برای انتخاب شده باشد تا با ساخت csv به ازای هر تست، آن فایل به همراه دیکشنری ذکر شده برای تجمیع و ساخت دو فایل merged.csv و merged_info.csv ارسال شود در کنار این فایل ها یک ورژن آنالیز شده از آنها نیز ایجاد می‌شود.

```
def mrbench_agent(config_params, config_file, config_output):
data = {**cosinfo_data, **data_swift, **data_workload, **ring_item}
if 'csv' in run_status_reporter:
     dashborad_images_dict, all_hosts_csv_dict, timeVariable =
status_reporter.main(metric_file=None, path_dir=result_dir,
time_range=f"{cosbench_data['start_time']},{cosbench_data['end_time']}",
img=imgFlag)


for group,csv_path in all_hosts_csv_dict.items():
     merged_file = analyzer.merge_csv(csv_file=csv_path,
output_directory=f"{result_dir}/analyzed",
mergedinfo_dict={**mergedinfo_data,**run_time},
merged_dict={**run_time,**merged_data,**mergedinfo_data})



if merged_file:
     analyzer.main(merge=False, analyze=True, graph=False,
csv_original=merged_file, output_directory=f"{result_dir}/analyzed",
selected_CSVs=None, keep_column=True)
```

- بخش چهارم تابع: در این بخش به ازای هر تست ابزار monstaver فراخوانی می‌شود و از اطلاعات تست بکاپ گرفته می‌شود و خروجی آن که آدرس دایرکتوری بکاپ است دریافت می‌شود. در آخر تابه نیز ۵ خروجی نهایی return می‌شوند که شامل first_start_time, last_end_time اولین زمان شروع و آخرین زمان پایان تست ها و workload برای ارسال به تابع monstaver_agent برای گرفتن یک بکاپ کلی از تمام بازه زمانی تست ها و backup_to_report که آدرس فایل بکاپ است برای ارسال به تابع report_recorder_agent و دیکشنری های dashborads_dict و all_hosts_csv_dict که شامل همه فایل های csv ساخته شده و همچنین داشبورد های دریافت شده از status_reporter هستند و باید به report_recorder_agent ارسال شوند.

```
def mrbench_agent(config_params, config_file, config_output):
if run_monstaver != 'none':
     if run_monstaver == 'backup,info':
          backup_to_report =
monstaver.main(time_range=f"{cosbench_data['start_time']},{cosbench_data['
end_time']}", inputs=[result_path,config_file,kara_config_files],
delete=False, backup_restore=None, hardware_info=None, software_info=None,
swift_info=None, influx_backup=True)


return first_start_time, last_end_time, backup_to_report,
all_hosts_csv_dict, dashborads_dict
```

۴ ـ تابع status_reporter_agent: این تابع واسط ابزار status_reporter است و در ورودی خود بخش مربوط به این ابزار را از فایل سناریو دریافت می‌کند.

در این قسمت لیستی از بازه های زمانی مورد نیاز برای دریافت گزارش از سناریو خوانده می‌شود و پس از آن در صورت نیاز به analyze و report بخش های مربوط به هرکدام خوانده شده و به تابع main دو ابزار analyzer و report_recorder ارسال می‌شوند تا برای برای هر گزارش سندی تولید شود و تحلیل و تجمیع روی csv هر یک انجام شود.

```
def status_reporter_agent(config_params):
if time_list:
        for time in time_list:
               # read file or list of time range
               if daily_report is True:
                      # make report in kateb
               if analyze_csv is True:
                      # make analyze and merge for csv
return dashborad_images_dict, all_hosts_csv_dict, time_list
```

در آخر خروجی تابع شامل لیست بازه های زمانی و دیکشنری های dashborads_dict و all_hosts_csv_dict که شامل همه فایل های csv ساخته شده و همچنین داشبورد های دریافت شده از status_reporter هستند و باید به report_recorder_agent ارسال شوند.

۵ ـ monstaver_agent: این تابع واسط ابزار monstaver است و در ورودی خود بخش مربوط به این ابزار را از فایل سناریو و خود فایل سناریو و لیست بازه های زمانی در تابع قبلی و همچنین اولین زمان شروع و آخرین پایان را از تابع mrbench_agent برای بکاپ گرفتن از دسته تست ها دریافت می‌کند.

در این قسمت لیست بازه های بکاپ یا از ورودی یا از فایل سناریو خوانده شده و عملیات بکاپ اجرا می‌شود اگر حالت batch_mode فعال بود از زمان اول و آخر که دریافت شده استفاده می‌کند. و در انتها مسیر داریکتوری بکاپ return می‌شود.

```
def monstaver_agent(config_params, config_file, start_time, end_time,
time_list):
if backup_time_list:
        for time in backup_time_list:
            # run monstaver main for each time
elif batch_mode:
 # run monstaver main for first start time and last end time
return backup_to_report
```

۶ ـ status_analyzer_agent: این تابع واسط ابزار analyzer است و در ورودی خود بخش مربوط به این ابزار را از فایل سناریو دریافت می‌کند.

در این بخش ورودی های مورد نیاز تابع main ابزار analyzer از سناریو خوانده شده و به صورت آرگومان های ورودی برای عملیات جمیع و تحلیل به صورت مجزا استفاده می‌شوند.

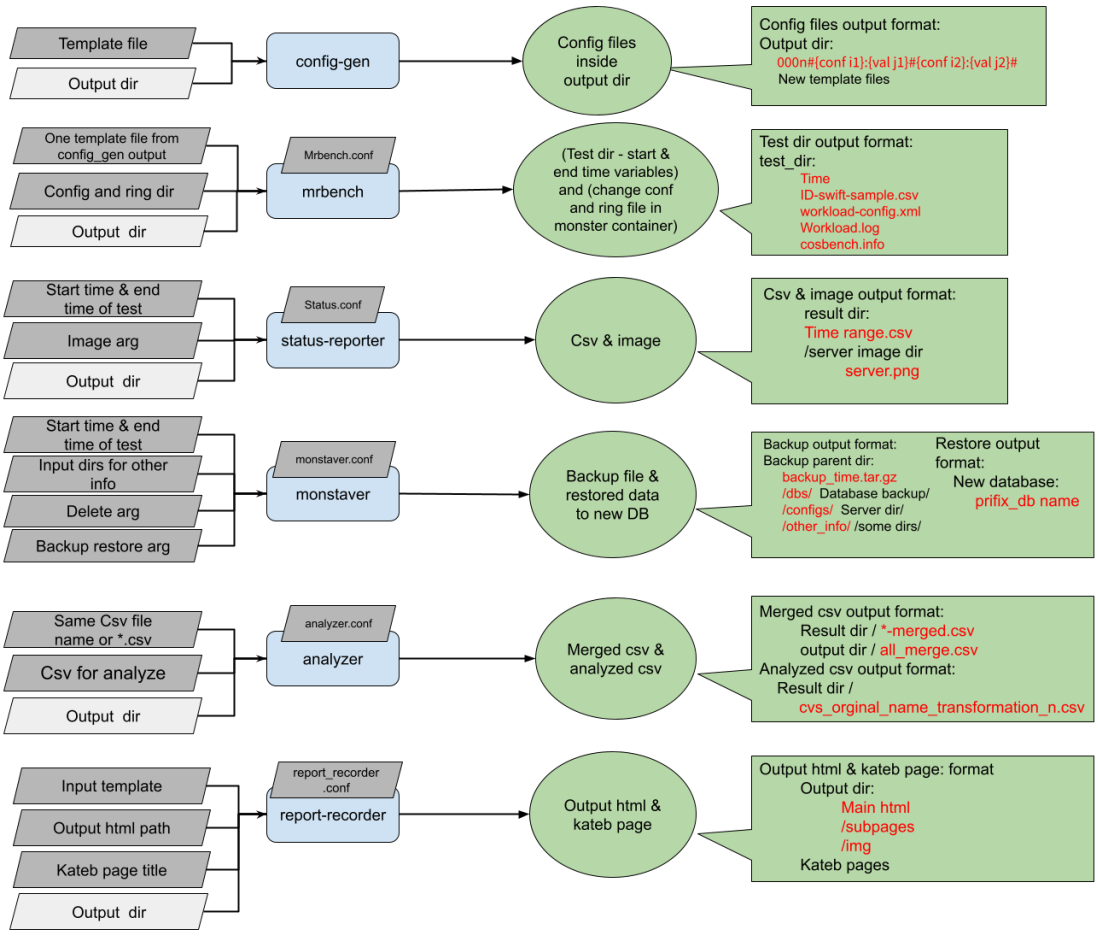
```
def status_analyzer_agent(config_params):
if merge:
        analyzer.main(merge=True, analyze=False, graph=False,
csv_original=None, output_directory=result_dir, selected_CSVs=merge_csv,
keep_column=None)
if analyze:
        analyzer.main(merge=False, analyze=True,
graph=make_analyzed_graph, csv_original=analyze_csv,
output_directory=result_dir, selected_CSVs=None,
keep_column=keep_source_columns)
```

۷ ـ report_recorder_agent: این تابع واسط ابزار report_recorder است و در ورودی خود بخش مربوط به این ابزار را از فایل سناریو و مسیر دایرکتوری بکاپ و دیکشنری داشبورد ها را دریافت می‌کند.

پس از دریافت ورودی ها تابع main ابزار فراخوانی می‌شود و آرگومان های مورد نیاز از سناریو دریافت و به آن ارسال می‌شوند.

```
def report_recorder_agent(config_params, backup_to_report,
dashborads_dict):

if sw_template or hw_template or monster_test_report is True:

        report_recorder.main()
```

۲-۳-۱-۳-۲- input/output : در این تصویر نوع خروجی و ورودی هر کی از ابزار ها شرح داده شده است.

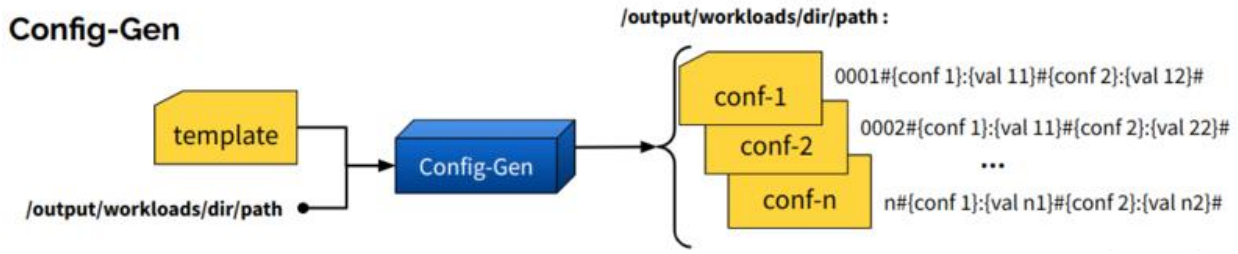


شکل ۶: kara i/o

۲-۳-۲- config_gen:

این ابزار وظیفه ایجاد کانفیگ های گوناگون با حالت های مختلف برای نرم افزار cosbench و swift را دارد.

این برنامه در دایرکتوری config_gen وجود داشته و ورودی آن یک فایل با فرمت های استاندارد cosbench و کانفیگ های swift است. این فایل ها مانند مثال های ذیل شخصی‌سازی شده اند و از هر یک از فایل ها با توجه به نوع تغییرات و نیاز کاربر می‌تواند چندین فایل با مقادیر متفاوت ایجاد کرد تا روند ساخت چندین فایل کانفیگ مختلف برای تست ها کارایی هیولا آسان تر و سریع تر شود.



شکل ۷:روند کار config-gen

۲-۳-۲-۱- ایجاد قالب cosbench

در زیر یک فایل اجرای بار کاری (workload) در فرمت xml برای نرم افزار cosbench وجود دارد که در بخش هایی برای استفاده در config_gen و ایجاد فایل های مختلف از روی آن شخصی‌سازی شده است.

```
<?xml version="1.0" encoding="UTF-8" ?>
<workload name="swift-sample" description="sample benchmark for swift">
  <storage type="swift" />
  <auth type="swauth"
config="username=test:tester;password=testing;auth_url=http://0.0.0.0:8080/auth/v1.0"/
>
  <workflow>
    <workstage name="init">
      <work type="init" workers="13" config="cprefix=?1L5s;containers=r(1,13)" />
    </workstage>
    <workstage name="main">
      <work name="main" workers="#1{8,16}concurrency#" runtime="60">
        <operation type="write" ratio="100"
config="containers=r(1,13);cprefix=?1L5s;objects=s(1,100);sizes=c(#2{128,256}objSize#)
KB" />
      </work>
    </workstage>
    <workstage name="dispose">
      <work type="dispose" workers="13" config="containers=r(1,13);cprefix=?1L5s" />
    </workstage>
  </workflow>
</workload>
```

در ذیل بخشی از یک فایل قالب ورودی cosbench و اجرای workload نمایش داده شده است در آن بخش هایی وجود دارند که مشخص کننده متغییر های کانفیگ و تعداد عناصر آن است مانند تعداد size , worker و غیره. روند کار به این صورت است

که از مقادیر داخل براکت {۸,۱۶} و {۲۵۶,۱۲۸} لیستی تهیه می‌شود و در هر بار ساخت فایل جدید یکی از آنها برای مقدار آن عنصر قرار می‌گیرند و به بیان دیگر به ازای هر کدام از این مقادیر حلقه ساخت فایل یکبار تکرار می‌شود . اگر در بخش های مختلف بعد از # اول یک عدد بیرون از براکت ها نوشته شد نمایانگر همروندی میان لیست های تشکیل شده است اگر اعداد یکسان بودند مقادیر داخل لیست ها مختلف متناظر با یکدیگر در فایل جدید قرار می‌گیرند اگر یکسان نبودند متفاوت و غیر همروند . در مرحله دیگر بعد از براکت ها متن یا اسمی نوشته شده که نمایانگر اسم آن قسمت از کانفیگ است که بعدا از آن برای نامگذاری فایل های کانفیگ استفاده می‌شود **#1{8,16}concurrency#** و **#2{256,128}objSize#**

```
<work name="main" workers="#1{8,16}concurrency#" runtime="120">
<operation type="write" ratio="100"
config="containers=r(1,13);cprefix=?1L5s;objects=s(1,10000);sizes=c(#2{256,12
8}objSize#)KB" />
```

برای نمونه این بخش از فایل کانفیگ ورودی در خروجی خود ۴ فایل کانفیگ جدید با کانفیگ های متفاوت را با این نوع نامگذاری ایجاد می کند:

0001#concurrency:8#objSize:128#

```
<work name="main" workers="8" runtime="120">
<operation type="write" ratio="100"
config="containers=r(1,13);cprefix=?1L5s;objects=s(1,10000);sizes=c(128)KB"
/>
```

0002#concurrency:8#objSize:256#

```
<work name="main" workers="8" runtime="120">
<operation type="write" ratio="100"
config="containers=r(1,13);cprefix=?1L5s;objects=s(1,10000);sizes=c(256)KB"
/>
```

0003#concurrency:16#objSize:128#

```
<work name="main" workers="16" runtime="120">
<operation type="write" ratio="100"
config="containers=r(1,13);cprefix=?1L5s;objects=s(1,10000);sizes=c(128)KB"
/>
```

0004#concurrency:16#objSize:256#

```
<work name="main" workers="16" runtime="120">
<operation type="write" ratio="100"
config="containers=r(1,13);cprefix=?1L5s;objects=s(1,10000);sizes=c(256)KB"
/>
```

۲-۳-۲-۲- تغییرات بعد از کاراکتر ?

با استفاده از oprefix و cprefix می‌توان برای آبجکت ها و کانتینر ها اسامی قرار داد و آنها را دسته بندی کرد و یا مواردی یکتا از آنها ساخت، برای ساخت آنها می‌توان از کاراکتر های عددی و حروفی استفاده کرد، برای این کار می‌توان مقادیری را به جای اسم آن ها نوشت تا ابزار این کار را به صورت خودکار انجام دهد. برای مثال: ? 1L5s یعنی ساخت مقداری تصادفی با شماره id 1که اگر این شماره در جای دیگری استفاده شود همین مقدار نیز برای آن تکرار می‌شود .L یعنی طول متن اسم و عدد بعد از آن نمایانگر طول اسم و تعداد کاراکتر است و s به معنی نوع آن است که string بوده و اگر از d استفاده شود یعنی decimal و مقدار عددی.

```
<workstage name="init">
      <work type="init" workers="13"
config="cprefix=?1L5s;containers=r(1,13)" />
</workstage>
```

خروجی کانفیگ ذکر شده برای cprefix ساخته شده:

```
<work name="main" workers="16" runtime="60">
       <operation type="write" ratio="100"
config="containers=r(1,13);cprefix=FP9I4;objects=s(1,10000);sizes=c(128)KB"
/>
      </work>
```

۲-۳-۲-۳- ایجاد کانفیگ های swift

در مواردی که نیاز است فایل های کانفیگ swift تغییر پیدا کنند و نمونه های مختلف از آنها ساخته شود که می‌توان مانند مثال زیر عمل کرد.

نمونه فایل اولیه کانفیگ proxy-server در swift :

```
[DEFAULT]
bind_ip = 0.0.0.0
bind_port = 8080
workers = #1{4,8,16}#
user = swift

log_statsd_host = controller
log_statsd_port = 8125
log_statsd_default_sample_rate = 1.0
log_statsd_sample_rate_factor = 1.0
log_statsd_metric_prefix = swift
```

یکی از فایل ها ساخته شده از کانفیگ ذکر شده در بالا:

```
[DEFAULT]
bind_ip = 0.0.0.0
bind_port = 8080
workers = 4
user = swift

log_statsd_host = controller
log_statsd_port = 8125
log_statsd_default_sample_rate = 1.0
log_statsd_sample_rate_factor = 1.0
log_statsd_metric_prefix = swift
```

۲-۳-۲-۴- روش استفاده از ابزار:

آرگومان های ورودی:

-i فایل قالب ورودی

-o مسیر دایرکتوری خروجی

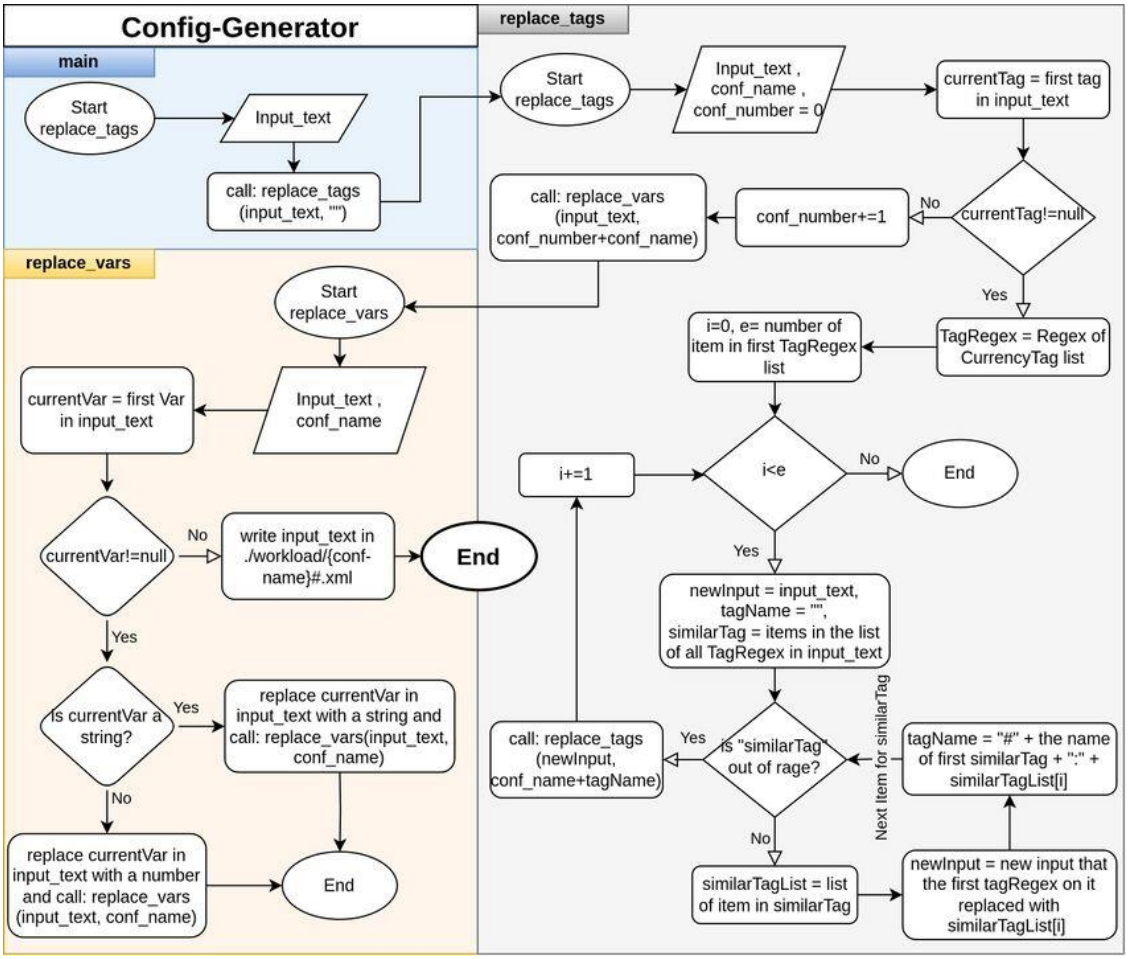
```
python3 config-gen.py -i input.txt -o <output dir>
```

۲-۳-۲-۵- توسعه ابزار config_gen:

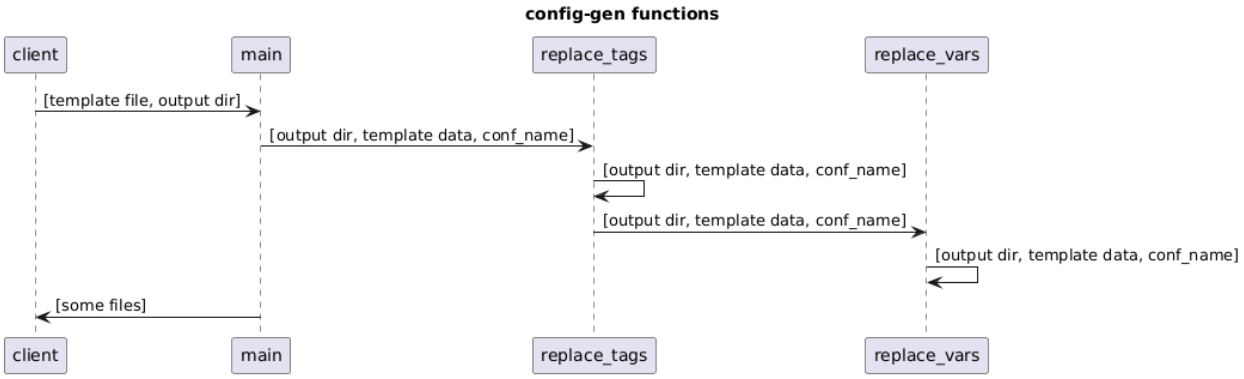
این ابزار وظیفه دارد یک فایل با هر نوع فرمت و ساختاری را در ورودی دریافت کند و با استفاده از تابع replace_tags قسمت هایی از فایل که دارای دو عدد کاراکتر " # " بود را پیدا کرده و مقدار اعداد بین این دو کاراکتر را به صورت همروند و غیر همروند در تمام قسمت های دارای # تغییر دهد.

پس از این با جستجو کاراکتر های " ? " علامت سوال تابع replace_vars مقادیری تصادفی و یا رندوم از نوع رشته با استفاده از اعداد و حروف می‌تواند بسازد که با طول های دلخواه ذکر شده در فایل ورودی بوده و این مقادیر می‌توانند به صورت همروند و یا غیر همروند با دیگر کاراکتر های علامت سوال تغییر کنند. ساختار کد دو تابع ذکر شده:

```
def replace_vars(input_text, conf_name, output_directory):
    currentVar = re.search("\?\d+L\d+[sd]", input_text)
    if currentVar:
        if currentVar.group()[-1] == 's':

replace_vars(input_text.replace(currentVar.group(),str(''.join(random.choices
(string.ascii_uppercase +
string.digits,k=int(currentVar.group().split('L')[1].split('s')[0])))),conf_
name, output_directory)
        else:

replace_vars(input_text.replace(currentVar.group(),str(''.join(random.choices
(string.digits,k=int(currentVar.group().split('L')[1].split('d')[0])))),conf
_name, output_directory)
    else:
        with open(os.path.join(output_directory, f"{conf_name}#"), 'w') as
outfile:
            outfile.write(input_text)
def replace_tags(input_text, conf_name, output_directory):
    global conf_number
    currentTag = re.search("#\d+{", input_text)
    if currentTag:
        TagRegex = currentTag.group() + "[^}]+}[^#]*#"
        for i in range(len(re.search(TagRegex,
input_text).group().split('}')[0].split(','))):
            newInput = input_text
            tagName = ""
            for similarTag in re.findall(TagRegex, input_text):
                similarTagList =
similarTag.split('{')[1].split('}')[0].split(',')
                newInput = re.sub(TagRegex, similarTagList[i], newInput, 1)
                tagName += ("#" + re.search("(?<=})[^#]*(?=#)",
similarTag).group() + ":" + str(similarTagList[i]))
            replace_tags(newInput, conf_name + tagName, output_directory)
    else:
        replace_vars(input_text, "{:04}".format(conf_number) + conf_name,
output_directory)
        conf_number += 1
```



شکل ۸: فلوچارت config_gen



شکل ۹: ساختار توابع config_gen

۲-۳-۳- mrbench:

این ابزار وظیفه اجرای کانفیگ های workload و ارسال آنها به cosbench را دارد. پس از اجرای تست و workload برای هر یک از آنها یک دایرکتوری یکتا بر اساس بازه زمانی اجرای آن در یک دایرکتوری والد ایجاد کرده و نتایج و اطلاعات به دست آمده از cosbench را در آن ذخیره می‌کند.

این ابزار توانایی تغییر فایل های گانفیگ swift و ring را نیز دارا است و می‌تواند فایل هایی با فرمتهای gz. و builder. و conf. را برای این کار از یک دایرکتوری که شامل فایل هایی با این فرمت ها است دریافت کند.

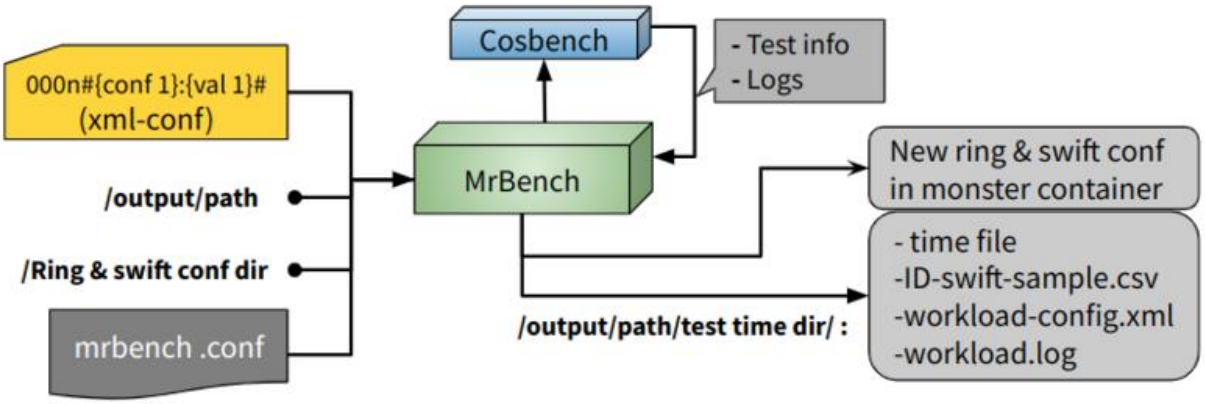
در مسیر /etc/kara/ فایل کانفیگ mrbench.conf وجود دارد که اطلاعات سرورهای هیولا و کانتینر های در آن موجود است و از این کانفیگ ها برای اتصال از طریق ssh و تغییر فایل های ring , swift استفاده می‌شود. در آخر پس از تغییرات در فایل های ring و swift برای اینکه تنظیمات جدید اعمال شود کانتینر های هیولا را restart می‌کند.

نکته: در اولین بار اجرای ابزار در صورتی که برای user ذکر شده در فایل کانفیگ ssh_key تعریف نشده باشد آنها را ایجاد و در سرور های هویلا کپی می‌کند.فقط درصورتی که ابزار فوق موفق به ساخت و کپی کلید ssh نشد دستورات زیر را به ازای تمام سرور های هویلا اجرا کنید.

```
su user
ssh-keygen (make sure new key just for new user and his .ssh directory)
ssh-copy-id -p <port> <user>@ip
```

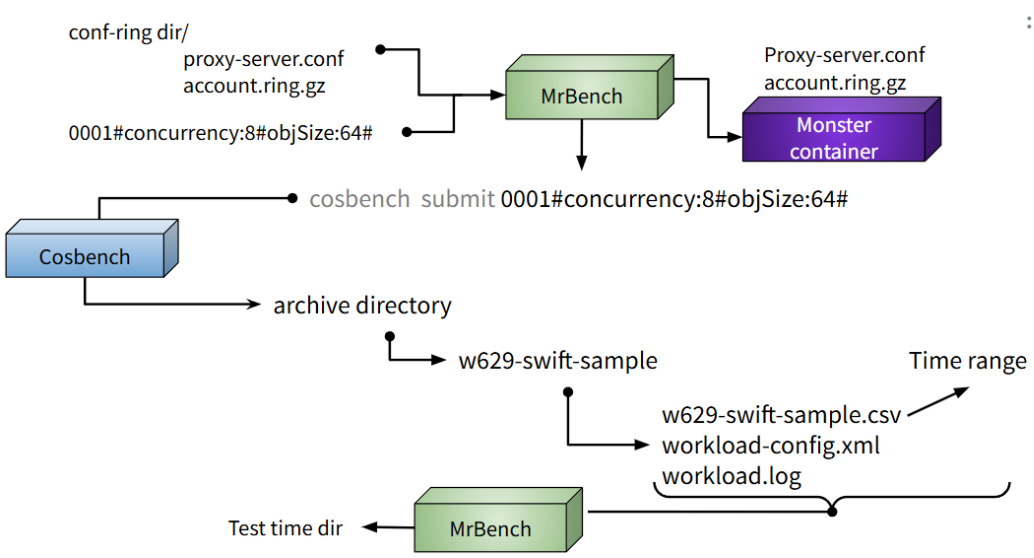
۲-۳-۳-۱- شرح فایل کانفیگ و اجرای هر بخش : در فایل کانفیگ این ابزار فقط نیاز به ذکر نام کانتینر های هیولا و اطلاعات ssh سرور های میزبان آن است.

```
monster:

   r1z1s1:     # monster container name
         ssh_user: user     # user in host
         ip: 0.0.0.0     # host ip
         ssh_port: 22     # host port

   r2z2s2:
         ssh_user: user
         ip: 0.0.0.0
         ssh_port: 22
log:
   level: info  # values = debug - info - warning - error - critical
```



شکل ۱۰: روند کار *mrbench*

مثال:



شکل ۱۱: مثال *mrbench*

۲-۳-۳-۲- روش استفاده از ابزار:

آرگومان های ورودی:

-i فایل تمپلیت workload ورودی به همراه آدرس کامل آن

-o مسیر خروجی

-cr دایرکتوری که شامل کانفیگ های swift و ring می‌باشد

```
python3 mrbench.py -i <path to test config file> -o <output dir> -cr <path to
config_ring dir>
```

۲-۳-۳-۳- توسعه ابزار mrbench :

این ابزار از ۳ تابع اصلی به همراه main تشکیل شده است که هریک در ذیل شرح داده شده اند.

۱ـ generate_and_copy_key: این تابع وظیفه ایجاد و کپی کردن ssh_key را از سرور اجرا کننده کارا بر روی سرور های هویلا را دارد. در ورودی اطلاعات سرور ها را از فایل کانفیگ که توسط تابع read_yaml_and_generate_keys خوانده شده است را دریافت می‌کند.

```
def generate_and_copy_key(username, ip, port, server_name):
if ssh key and dir not exist:
        # make .ssh dir
        # make key
        subprocess.run(['sudo', '-u', username, 'ssh-keygen', '-t', 'rsa', '-
b', '2048', '-f', str(ssh_key_path), '-N', ''], check=True)
        print(f"SSH key generated for {username} at {ssh_key_path}")

# Check if the public key already exists on the remote server
    try:
        # SSH to the remote server and check for the public key in
authorized_keys
        check_key_command = f"ssh -p {port} {username}@{ip} 'grep -q
\"{public_key}\" ~/.ssh/authorized_keys'"
        subprocess.run(check_key_command, shell=True)

# Use sshpass to provide the password non-interactively
subprocess.run(['sshpass', '-p', password, 'ssh-copy-id', '-p', str(port), '-
i', public_key_path, f"{username}@{ip}"], check=True)
```

۲ـ copy_swift_conf: این تابع وظیفه کپی کردن فایل های جدید ring و swift_conf را در سرور های هیولا دارد ، با این کار قبل از اجرای تست های هیولا کانفیگ ها تغییر می‌کنند و تست در حالت های مختلف اجرا می‌شود. برای افزایش کارایی و بهبود سرعت اجرا این تابع از ویژگی multi threading بهره می‌برد و برای این کار از تابع conf_ring_thread برای انجام فرآیند های ذکر شده استفاده می‌کند.

این تابع در ورودی خود دیکشنری شامل اسم فایل های کانفیگ و رینگ به همراه آدرس آنها را دریافت می‌کند.

swift_configs[filename] = file path

```
def copy_swift_conf(swift_configs):

ring_dict = {}
data_loaded = load_config(config_file)

with concurrent.futures.ThreadPoolExecutor() as executor:

            # run in multithread
```

```
            future = executor.submit(conf_ring_thread, swift_configs, port,
user, ip, container_name, key_to_extract)
            futures.append(future)

return ring_dict
```

```
def conf_ring_thread(swift_configs, port, user, ip, container_name,
key_to_extract):

ring_dict = {}

if filename.endswith(".gz") or filename.endswith(".builder"):

        # copy ring files and make ring_dict

elif filename.endswith(".conf"):

        # copy swift conf files

if all_scp_file_successful is True:

        #if all process successful restart containers

return ring_dict
```

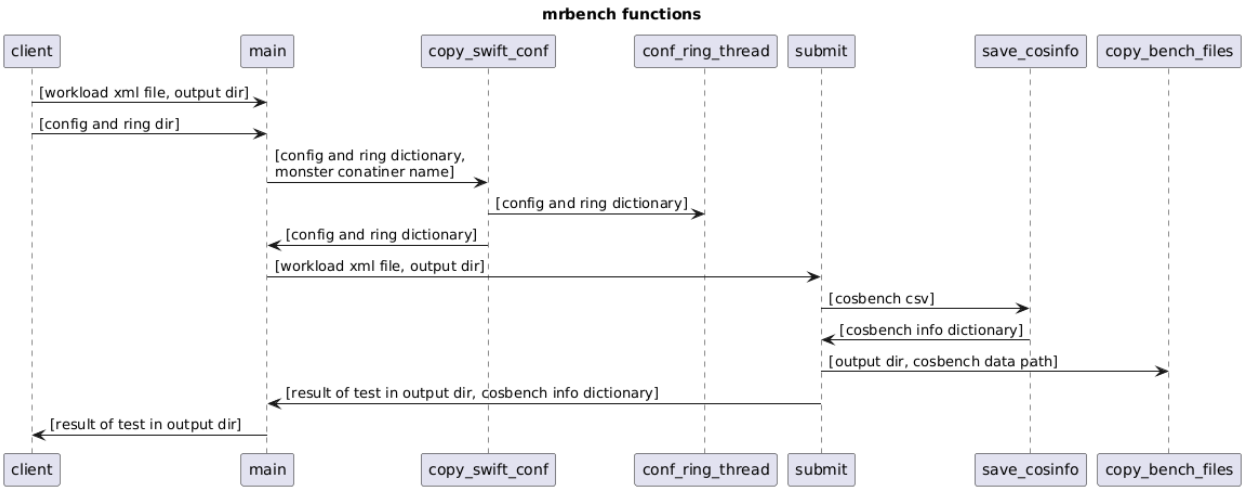
پس از اجرای موفق و کپی شدن فایل ها برای اعمال آنها کانتینر های هیولا ری استارت می‌شوند و در آخر دیکشنری ring_dict در توابع return می‌شود که شامل اطلاعات فایل های رینگ به فرمت ذیل است.

```
if "account" in filename:

    ring_dict['account'] = account.builder

elif "container" in filename:

    ring_dict['container'] = container.builder

else:
    ring_dict['object'] = object.builder
```

۳ـ submit: این تابع وظیفه اجرای workload را با استفاده از نرم افزار cosbench دارد. پس از اجرای تست و در یافت نتایج زمان شروع و پایان تست را از csv ساخته شده توسط cosbench استخراج و دایرکتوری برای تست با نامی مشابه بازه زمانی آن ایجاد می‌کند و فایل های مختص هر تست را در آن قرار می‌دهد. این تابع در ورودی فایل xml تست و مسیر خروجی نهایی را دریافت می‌کند.

```
def submit(workload_config_path, output_path):

# run workload with cosbench

# extract cosbench csv data and copy result files
if os.path.exists(archive_file_path):

        cosbench_data =
save_cosinfo(f"{archive_path}{archive_workload_dir_name}/{archive_workload_di
r_name}.csv")

copy_bench_files(archive_path, archive_workload_dir_name, result_path)
```

شکل ۱۲: فلوچارت *mrbench*



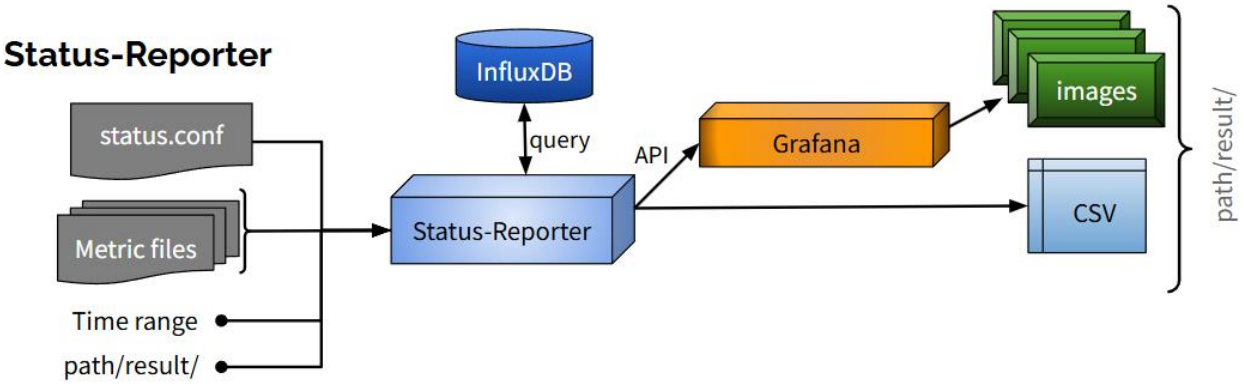
شکل ۱۳: ساختار توابع *mrbench*

۲-۳-۴- status_reporter:

این ابزار وظیفه ارسال query به influxdb را دارد و بر اساس measurement یا متریک های مورد نیاز جواب را دریافت کرده و خروجی به دو صورت فایل csv و فایل تصویر گراف تولید می کند.

این ابزار در دایرکتوری status_reporter و تنظیمات آن در مسیر /etc/kara/ در فایل status_reporter.conf موجود هستند. فایل های متریک آن نیز در مسیر اصلی خود ابزار در دایرکتوری metrics قرار دارد.

این ابزار برای ساخت تصاویر از نرم افزار grafana استفاده می‌کند. برای این کار باید یک داشبورد گرافانا با فرمت .json که از قبل export شده است را در دایرکتوری jsons این ابزار قرار داده شوند، سپس ابزار با استفاده از API گرافانا داشبورد را import می‌کند و تصاویر پنل های آن را به ازای هر سرور هیولا به صورت مجزا ذخیره سازی می‌کند. پس از این کار، یک نمونه تصویر تجمیع شده از تمام تصاویر هر داشبور به ازای هر سرور نیز ایجاد می‌کند که در مراحل بعدی در اسناد کاتب استفاده می‌شوند.



شکل ۱۴: روند کار status_reporter

۲-۳-۴-۱- شرح فایل کانفیگ و اجزای هر بخش:

فایل status_reporter.conf از ۳ بخش اصلی تشکیل شده است که هر کدام در ادامه توضیح داده شده اند:

۱- بخش اطلاعات دیتابیس و گرافانا: شما می‌توانید این بخش از کانفیگ را به ازای هر یک از سرور های mc تکرار کنید . status_reporter پس از دریافت اطلاعات دیتابیس، سرور mc و نرم افزار گرافانا' تصاویر و csv هایی از بازه های زمانی داده شده به آن را دریافت می‌کند.

برای دریافت گزارش در قالب عکس نیاز است ابتدا از داشبورد مورد نظر در گرافانا یک فایل json استخراج شود و در دایرکتوری jsons ابزار ذخیره شود، پس از آن با استفاده از API ، داشبورد های موقت در گرافانا import شده و تصاویری از پنل های درون آن با استفاده از پلاگین image renderer خود گرافانا به ازای هر host استخراج می‌شود و یک یا چند نمونه تجمیع شده از عکس ها نیز با توجه به نیاز کاربر تولید می شود. همزمان با این فرآیند تولید عکس، نتایج query هایی که مستقیما به دیتابیس ارسال شده اند به صورت فایل های csv برای هر سرور هیولا ساخته شده و در دایرکتوری هایی با نام زمان گزارش ذخیره می‌شوند.

```
influxdbs:
   MC:
      grafana_dashboards:
         remove_dashboards: True     # remove temporary dashboard of images
         time_variable: 10s     # time variable in dashboards
         dashboards_name:
               - Performance_Overview
               - Partial_Monitoring
               #- custom
         custom_panels:  # needs to select custom dashbord in dashboards
section
               - network
               - cpu
         report_images:
           # panels_per_row: 2
           # panels_per_column: 3
           # max_panels: 9      # panels_per_row ingnored if set max_panels
           # panel_width: 800   # size in pixel
           # panel_height: 400  # size in pixel
      grafana_api_key: "add your API-key here"
      grafana_port: 3000
      grafana_ip: 0.0.0.0 # ip of grafana host
      influx_port: 8086
      influx_ip: 0.0.0.0 # ip of influxdb host
      databases:
         opentsdb:
            hosts:
               mc:
                  - "mc1"
                  - "mc2"
               monster:
                  - "r2z2s2-controller"
                  - "r1z1s1-controller"
```

نکته: در داشبورد هایی که توسط کاربر ساخته شده اند و به این ابزار داده می‌شوند باید حتما از متغیر هایی با این نام ها استفاده شود تا فرآیند استاندارد سازی داشبورد ها به درستی انجام شود.

(هاست ایز) hostIs = نمایش و انتخاب سرور ها

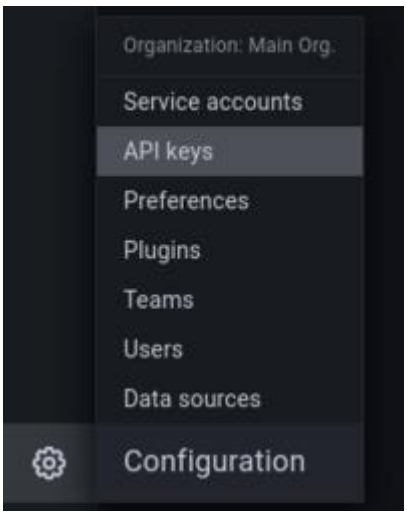
timeVariable = تایم فریم

شرح بخش کانفیگ ذکر شده:

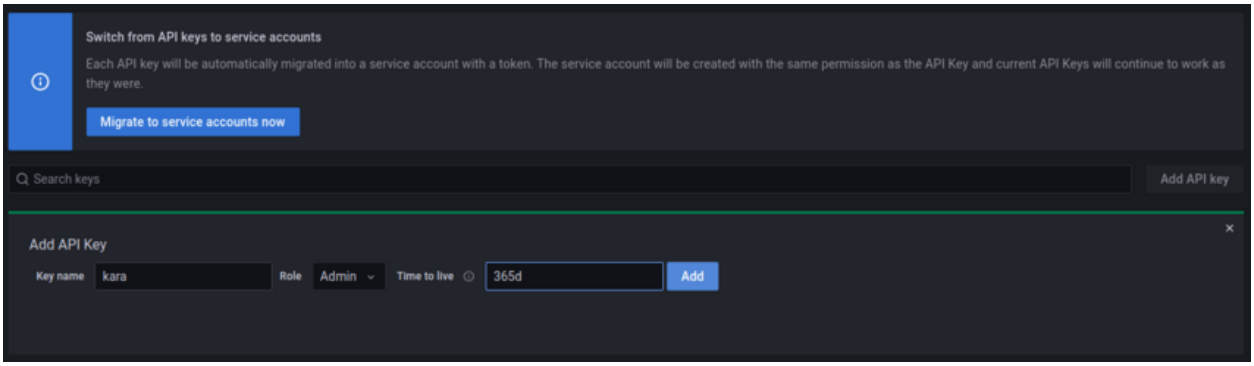
- grafana_dashboards:
  - remove_dashboards: برای حذف کردن یا نکردن داشبورد موقت import شده به گرافانا است.
  - time_variable: متغیر بازه های زمانی در گراف پنل ها است که مقدار آن باید در داشبورد ها تعریف شده باشد.

- dashboards_name: لیستی از اسامی داشبورد های موجود در دایرکتوری .jsons اگر این داشبورد ها از قبل در گرافانا وجود داشته باشند به انتهای اسامی آنها یک شماره اضافه شده و سپس import می‌شوند.
- custom_panels: این ابزار این قابلیت را نیز دارد که داشبورد هایی شخصی سازی شده تولید کند به این صورت که اگر یک فایل json داشبورد خالی از پنل استخراج کنید و نام آن را به customتغییر دهید می‌تواند فایل های json استخراج شده از دیگر پنل ها را که در همان مسیر jsonsقرار دارد به داشبورد خالی اضافه کند و عکس را از آن ها دریافت کند.
- report_images: در ابتدای این بخش گفته شد که پس از دریافت تصاویر برای هر داشبورد و سرور هیولا یک یا چند عکس تجمیع شده مشابه داشبورد از آنها ساخته می‌شود در این قسمت از فایل کانفیگ می‌توان ابعاد عکس تجمیع شده و تعداد کل تصاویر پنل درون آن و تعداد پنل های در سطر و ستون آن را مشخص کرد.

- grafana_api_key: برای ارسال و دریافت اطلاعات به گرافانا نیاز است از API آن استفاده شود برای این کار می‌توان مشابه تصاویر زیر یک API-key در گرافانا ساخت. پس از ایجاد API-key با رول admin و نام و تاریخ اعتبار دلخواه کد نمایش داده شده را در کانفیگ status_reporter قرار دهید.



مرحله اول



مرحله دوم

- grafana_port و grafana_ip: در این دو بخش باید ip , port سرور میزبان گرافانا ذکر شود.
- influx_port و influx_ip: در این دو بخش باید ip , port سرور میزبان influxdb ذکر شود.
- Databases: در این بخش نیز لیستی از دیتابیس های موجود و سرور های هویلا در گروه بندی های مختلف موجود در هر دیتابیس قرار دارد. می‌توان این بخش را به ازای هر دیتابیس متفاوت تکرار کرد.

۲- بخش اطلاعات فایل های متریک:

بخش دوم کانفیگ مربوط به فایل های measurement یا metric است در این قسمت عملیات ریاضی هر فایل و آدرس آن ذکر شده و امکان تغییر آنها در سوئیچ های ورودی یا آرگومان برنامه نیز وجود دارد. درون هر فایل لیستی از متریک های ارسال شده توسط netdata و ذخیره شده در influxdb وجود دارد که امکان کامنت کردن متریک های غیر ضروری نیز برای آنها فراهم شده.

فرمت نامگذاری متریک ها در فایل های موجود باید به این شکل باشدnetdata.n.n.n :

این ابزار امکان تکمیل کردن اسم متریک ها را با استفاده از regex نیز دارد برای این کار کافی است اسم متریک مورد نظر را در فرمت های مثال زده شده بنویسید.

```
/*.netdata.system.cpu/
/netdata.disk_ops.sd[abc].reads/
```

بخش ذکر شده درون فایل کانفیگ:

```
metrics:

  sum:
    path: ./../status_reporter/metrics/sum_metric_list.txt
  mean:
    path: ./../status_reporter/metrics/mean_metric_list.txt
  max:
    path: ./../status_reporter/metrics/max_metric_list.txt
  min:
    path: ./../status_reporter/metrics/min_metric_list.txt
```

نکته: در صورتی که ابزار در فایل سناریو اجرا شود نیازی به تنظیم این بخش نیست و مقادیر زمان و خروجی از سناریو دریافت می‌شود.

۳ – بخش اطلاعات زمانی و خروجی:

در این قسمت امکان تعریف margin های زمانی از اول و آخر بازه ریپورت به منظور دقیقتر بودن خروجی وجود دارد و واحد آن به ثانیه است.

در قسمت آخر بازه زمانی گزارش قرار دارد که شامل زمان شروع و پایان گزارش است و فرمت آن tehran timestamp بوده و در صورتی که زمان دقیق و مشخصی برای بازه گزارش وجود نداشته باشد این امکان وجود دارد که از زمان حال حاضر اجرای برنامه تا یک بازه زمانی ماقبل آن نیز گزارش گرفته شود مشابه این فرمت ها:

(از زمان حال حاضر منهای ۲ ساعت قبل برای زمان شروع و زمان حال حاضر برای پایان) now-2h , now

(از زمان حال حاضر منهای ۲ روز قبل برای زمان شروع و زمان حال حاضر برای پایان) now-2d , now

output_path: مسیر خروجی برنامه

```
time:
  start_time_sum: 10  # increase your report start time
  end_time_subtract: 10  # decrease your report end time
  time_range: 2024-09-08 12:00:00,2024-09-08 12:20:00 # can take two format
"now-nh,now-nh" or timestamp "Y-M-D h-m-s,Y-M-D h-m-s"

output_path: /path/to/results
```

۲-۳-۴-۲- روش های استفاده از ابزار:

در صورتی که نیاز می‌توان این موارد را در آرگومان ورودی به ابزار داد در غیر این صورت از فایل کانفیگ فراخوانی می‌شوند.

آرگومان های ورودی:

-m لیستی از فایل های متریک path to metric file 1,path to metric file 2,path to metric file 3 ,...

-t بازه زمانی تست'start time , end time'

-o مسیر خروجی

--img عملیات ساخت عکس ها

```
python3 status_reporter.py --img
```

در آخر نتایج در دایرکتوری داده شده به برنامه در دایرکتوری به نام بازه زمانی گزارش که در کانفیگ داده شده به صورت گراف هایی برای هر سرور هیولا و یک فایل csv برای هر گروه از سرور ها ذخیره می شود که اسم آن نیز بازه زمانی گزارش است مشابه تصویر پایین.

```
2024-09-08_12-00-00__2024-09-08_12-20-00/
├── mc_2024-09-08_12-00-00__2024-09-08_12-20-00.csv
├── mc_r1z1s1-controller-images
│   ├── Partial_Monitoring_CPU.png
│   ├── Partial_Monitoring_dashboard__1.png
│   ├── Partial_Monitoring_Disk_BW.png
│   ├── Partial_Monitoring__etc_localtime_BW.png
│   ├── Partial_Monitoring_GET_HEAD_Account_Container_Object.png
│   ├── Partial_Monitoring_Memory_Paged_from_to_disk.png
│   ├── Partial_Monitoring_Memory_Utilization.png
│   ├── Partial_Monitoring_Network_Bandwidth.png
│   ├── Partial_Monitoring_Per_Disk_BW.png
│   └── Partial_Monitoring_Per_Disk_Throughput.png
├── monster_2024-09-08_12-00-00__2024-09-08_12-20-00.csv
└── monster_r2z2s2-controller-images
    ├── Partial_Monitoring_CPU.png
    ├── Partial_Monitoring_dashboard__1.png
    ├── Partial_Monitoring_Disk_BW.png
    ├── Partial_Monitoring__etc_localtime_BW.png
    ├── Partial_Monitoring_GET_HEAD_Account_Container_Object.png
    ├── Partial_Monitoring_Memory_Paged_from_to_disk.png
    ├── Partial_Monitoring_Memory_Utilization.png
    ├── Partial_Monitoring_Network_Bandwidth.png
    ├── Partial_Monitoring_Per_Disk_BW.png
    └── Partial_Monitoring_Per_Disk_Throughput.png

2 directories, 22 files
```

شکل ۱۵: ساختار خروجی status_reporter

۲-۳-۴-۳- توسعه status_reporter: این ابزار از ۴ تابع اصلی و ۳ تابع برای تبدیل های زمانی تشکیل شده است که توابع اصلی به شرح ذیل می‌باشند.

۱ـ get_report: اصلی ترین تابع ابزار است، وظیفه دارد از طریق query با دیتابیس در ارتباط باشد و نتایج به دست آمده از آن را در فایل csv با ساختاری مشخص ذخیره کند و در کنار این مورد در صورتی که نیاز به ساخت تصاویر نیز باشد توابع مورد نیاز آن را فراخوانی کرده و ورودی های مورد نیاز را به آنها بدهد.

این تابع در ورودی خود داده های استخراج شده از فایل کانفیگ و فایل های متریک و مسیر خروجی به همراه بازه زمانی مورد نیاز در query و آرگومان ساخت تصاویر را دریافت می‌کند.

```
def get_report(data_loaded, metric_file, path_dir, time_range, img=False):

for mc_server, config in data_loaded.get('influxdbs', {}).items():
        # load data of each server
        if img:
                # load data of making image
        for db_name, db_data in config.get('databases', {}).items():
                # each database
                for group_name, hostIsList in db_data['hosts'].items():

        dashborad_images_dict[f'{start_time_csv}__{end_time_csv}'][group_name]
= {}
                        # each host
                        if img:
                                # make image for each server and database

                                # make image from Grafana
                                images_path_dict = images_export()

                                img_dashboard_dict =
dashboard_maker_with_image()

dashborad_images_dict[f'{start_time_csv}__{end_time_csv}'][group_name][host_n
ame] = img_dashboard_dict # make dictionary inside dictionary
                    output_csv_str.append(host_name)

                        for metric_file, metric_operation in
metric_operation_mapping.items():

                                # send query for each metric and server and
database

# save query result to csv file

if dashboard_rm is True and img:
            # remove temp dashboard
             remove_dashboard(grafana_url, api_key, dashboard_data_dict)

return dashborad_images_dict, all_hosts_csv_dict
```

پس از ساخت تصاویر یک دیکشنری با نام dashborad_images_dict در خروجی تابع قرار می‌گیرد که دارای ساختاری مانند ذیل بوده و برای دیگر ابزار های کارا استفاده می‌شود.

dashborad_images_dict[f'{start_time_csv}__{end_time_csv}'][group_name][host_name] = {dashboard_name : list of images}

برای فایل های csv نیز دیکشنری all_hosts_csv_dict ساخته می‌شود که شامل گروه بندی سرور ها و مسیر فایل csv نهایی هر کدام از سرورها است.

all_hosts_csv_dict[group_name] = output_csv_pat

۲ـ dashboard_import: این تابع وظیفه آپلود لیستی از داشبورد های گرافانا را دارد. به این صورت که ابتدا بررسی می‌شود داشبورد تکراری است یا نه سپس اگر تکراری بود نام و uid و id آن را تغییر می‌دهد و سپس آپلود می‌کند. این تابع در ورودی خود لیستی از داشبورد ها درون فایل کانفیگ و API گرافانا و url آن و در حالاتی پنل های جدید را برای داشبورد custom در یافت می‌کند.

```
def dashboard_import(dashboards_json, api_key, grafana_url,
customized_panles):

dashboard_data_dict = {}

for dashboard_org_name in dashboards_json:

        if "custom" in dashboard_org_name:
                # make custom dashboard with new panels

        if dashboard_data["title"] in existing_names:
                # change name and uid and id of new dashboard

        # import dashboard
        response = requests.post(grafana_url+"/api/dashboards/db",
headers=headers, json=payload)

return dashboard_data_dict
```

در آخر پس از انجام همه فرآیند های آپلود داشبورد یک دیکشنری با ساختاری شبیه مورد زیر ایجاد می‌کند.

dashboard_data_dict[dashboard_name] = [dashboard_org_name, dashboard_uid_img, dashboard_data]

۳ـ images_export: این تابع وظیفه استخراج تصاویر از پنل های داشبورد های آپلود شده را دارد و این کار را با استفاده از پلاگین image renderer خود گرافانا انجام می‌دهد. در ورودی اطلاعات گرافانا و دیکشنری ساخته شده توسط تابع قبلی و اطلاعات هاست ها و گروه های آنها و ابعاد تصاویر خروجی را از فایل کانفیگ دریافت می‌کند.

```
def images_export(dashboard_data_dict, api_key, grafana_url, start_time_utc,
end_time_utc, output_parent_dir, hostList, panel_width, panel_height,
time_variable, group_name):

images_path_dict = {}

for dash_name, values in dashboard_data_dict.items():

        dashboard_org_name = values[0]

        dashboard_uid_img = values[1]

        dashboard_data = values[2]

        for panel in panels:

                for host in hostList:

                # use API to export images

                 hostAPIstr+=f"{var_host}={host}&"
            curl_api = (f'curl -o
{each_server_path}/{dashboard_org_name}_{panel_title}.png -H "Authorization:
Bearer {api_key}" "{grafana_url}/render/d-
solo/{dashboard_uid_img}/{dash_name}?orgId=1&{hostAPIstr}{var_time}={time_var
iable}&from={start_unix}&to={end_unix}&panelId={panel_id}&width={panel_width}
&height={panel_height}&tz={api_timezone}"')


return images_path_dict
```

در آخر پس از انجام همه فرآیند های ساخت تصویر یک دیکشنری با ساختاری شبیه مورد زیر ایجاد می‌کند.

images_path_dict[dashboard_org_name].append(image_path)

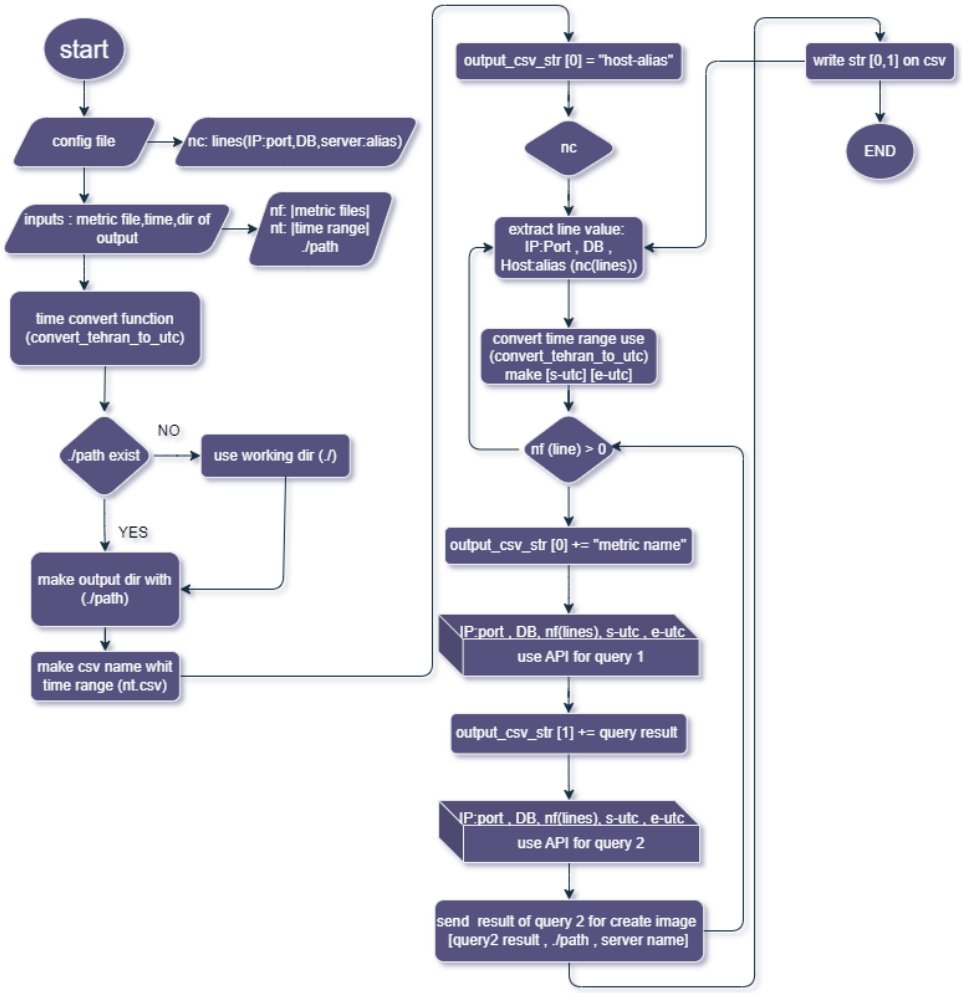
۴ـ dashboard_maker_with_image: این تابع عملیات تجمیع عکس های ساخته شده در یک عکس مانند داشبورد گرافانا را دارد. ابعاد و اندازه تصاویر از فایل کانفیگ دریافت می‌شود، در صورتی که تصویر نهایی بیش از اندازه دارای پنل بود و ابعاد زیادی داشت آن را به تصاویر کوچک تر تقسیم می‌کند. این تابع در ورودی خود دیکشنری ساخته شده در تابع قبلی که شامل آدرس فایل های تصاویر بود به همراه مسیر خروجی و ابعاد تصویر نهایی را دریافت می‌کند.

```
def dashboard_maker_with_image(images_path_dict, output_file,
panels_per_column, max_panels):

output_files_dict = {}

for dashboard_org_name, image_paths in images_path_dict.items():

       # Split the images into chunks

        image_chunks = [image_paths[i:i + max_panels] for i in
range(0, len(image_paths), max_panels)]

       for idx, chunk in enumerate(image_chunks):

            images = [Image.open(image_path) for image_path in chunk]

              # make images and calculate images size
              # Save the combined image with a unique name for each
chunk

            result_image_name =
f"{output_file}/{dashboard_org_name}_dashboard__{idx + 1}.png"

dash_list.append(result_image_name)

output_files_dict[dashboard_org_name] = dash_list
return output_files_dict
```

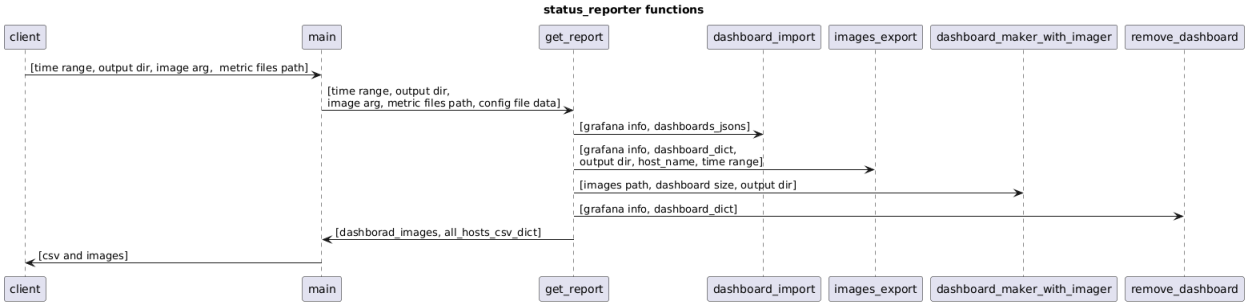
در آخر پس از ساخت تصاویر داشبورد مانند دیکشنری شامل اسامی داشبورد ها و لیستی از تصاویر موجود در هر کدام ایجاد می‌شود.

output_files_dict[dashboard_org_name] = dash_list

نکته : دیکشنری ها و تصاویر و csv های ساخته شده در این ابزار در report_recorder و analyzer استفاده می‌شوند.



شکل ۱۶: فلوچارت status_reporter



شکل ۱۷: توابع status_reporter

۲-۳-۵- monstaver:

این ابزار وظیفه دارد یک بازه زمانی دلخواه را از کاربر دریافت کند و در آن بازه زمانی از پایگاه داده influxdb یک نسخه پشتیبانی ایجاد کند.

در ادامه می‌تواند کانفیگ های swift که شامل ring و object و container و account هستند را نیز از هیولا دریافت به دایرکتوری نهایی محل بکاپ منتقل کند، از دیگر ویژگی های ابزار این است که در کنار موارد ذکر شده می‌تواند نتایج تست و گزارش که توسط دیگر ابزار ها تولید می‌شود را نیز همراه با بکاپ و کانفیگ ها قرار داده و آنها را پس از دسته بندی برای ذخیره سازی و جابجایی راحت تر فشرده سازی کند.

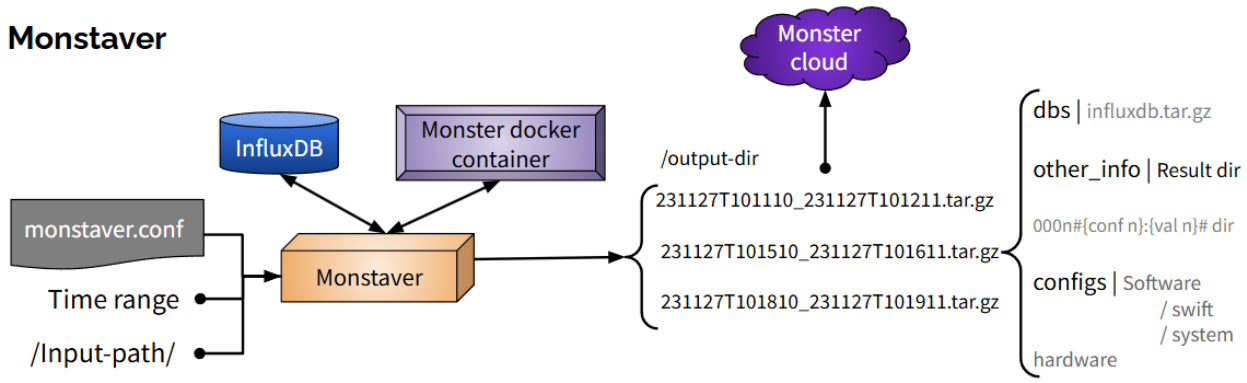
در صورت نیاز امکان restore کردن بکاپ های گرفته شده در influxdb برای آن نیز نظر گرفته شده است.

از دیگر ویژگی های این ابزار می توان به آپلود فایل فشرده شده نهایی بکاپ در کلاستر هیولا به عنوان مثال zdrive :اشاره کرد.

کانفیگ ها برنامه در مسیر /etc/kara/ در فایل monstaver.conf قرار دارد.

نکته: در اولین بار اجرای ابزار در صورتی که برای user ذکر شده در فایل کانفیگ ssh_key تعریف نشده باشد آنها را ایجاد و در سرور های هویلا و mc کپی می‌کند. فقط درصورتی که ابزار فوق موفق به ساخت و کپی کلید ssh نشد دستورات زیر را برای تمام سرور های هویلا و mc اجرا کنید.

```
su user
ssh-keygen (make sure new key just for new user and his .ssh directory)
ssh-copy-id -p <port> <user>@ip
```



شکل ۱۸: روند کار monstaver

۲-۳-۵-۱- شرح فایل کانفیگ و اجزای هر بخش:

کانفیگ این ابزار از دو بخش اصلی تشکیل شده. بخش اول برای بکاپگیری و بخش دوم برای بازیابی است. بخش بکاپگیری خود از سه قسمت تشکیل شده که به تفکیک هر کدام در ادامه توضیح داده شده اند:

۱- بخش بکاپ (قسمت default)

در این بخش اطلاعات پایه فرآیند بکاپگیری قرار دارد که هر کدام در ذیل توضیح داده شده اند:

- time , time_margin : بازه زمانی بکاپ و مارجین های زمانی برحسب ثانیه که به زمان شروع افزوده و از زمان پایان بکاپ کسر می‌شوند تا بازه زمانی دقیقتری در بکاپ داشته باشیم.
- input_path : لیستی از مسیر های دلخواه که نیاز هستند در فایل نهایی وجود داشته باشند مانند مسیر خروجی تست و گزارش.
- backup_output : محل خروجی برنامه در سرور اجرا کننده کارا.
- upload_to_monster : اطلاعات کلاستر هیولا برای آپلود فایل بکاپ در آن. در صورت نیاز تداشتن به این بخش گزینه آپلود را false کنید نیاز به کامنت کردن آن نیست. موارد زیر مجموعه این بخش همراه با مثال:
  - "**/token_url:** "https://api.zdrive.ir/auth/v1.0
  - "**public_url:** "https://api.zdrive.ir/v1/AUTH_user
  - "**username:** "user:user
- backup-options: در این بخش موارد انتخابی برای قرار گرفتن در فایل بکاپ نهایی وجود دارد که شامل اطلاعات سخت افزاری و نرم افزاری سرور هیولا و اطلاعات کانتینر هیولا و کانفیگ های swift و ring است.

```
default:

   # time,margin: start,end
   time: 2024-09-1 15:00:00,2024-09-01 15:02:00 # can take two format "now-
nh/d,now-nh/d" and "now-nh/d-now" or timestamp "Y-M-D h-m-s,Y-M-D h-m-s"
   time_margin: 10,10

   # some dir inside my local server
   input_paths:
         - /path/to/custom dir

   # output of all part in my local server
   backup_output: /tmp/influxdb-backup

   # monster storage info for upload backup
   upload_to_monster:
      upload: False
```

```
      token_url: # add your user token_url here
      public_url: # add ypur public_url here
      username: # add your username here
      password: # add your password here
      cont_name: kara # container name in monster

   # make backup from hardware/software/swift
   backup-options:
      hardware_backup: True
      software_backup: True
      swift_backup: True
```

۲- بخش بکاپ (قسمت swift)

در این قسمت اطلاعات سرورهای هیولا و نام کانتینر ها وجود دارد و از این کانفیگ ها برای اتصال با ssh و دریافت فایل های swift , ring و کانفیگ های نرم افزاری و سخت افزاری سرور های هیولا استفاده می شود.

```
swift:
   r1z1s1:    # monster container name
         ip: 0.0.0.0    # host ip
         ssh_port: 22    # host port
         ssh_user: user    # user in host
   r2z2s2:
         ip: 0.0.0.0
         ssh_port: 22
         ssh_user: user
```

۳- بخش بکاپ (قسمت db_sources)

در این قسمت اطلاعات mc که دارای کانتینر influxdb است به همراه mount point درون کانتینر که برای ذخیره سازی موقت بعضی از فایل ها استفاده می‌شود و اسامی دیتابیس های موجود در آن قرار دارد این بخش می‌تواند به ازای سرور های mc تکرار شود.

```
db_sources:

   MC:
      ip: 0.0.0.0
      ssh_port: 22
      ssh_user: user
      container_name: influxdb
      influx_volume: /var/lib/influxdb/KARA_BACKUP    # mount point inside
influxdb container
      databases:    # list of databases
            - opentsdb
```

۴- بخش بازیابی:

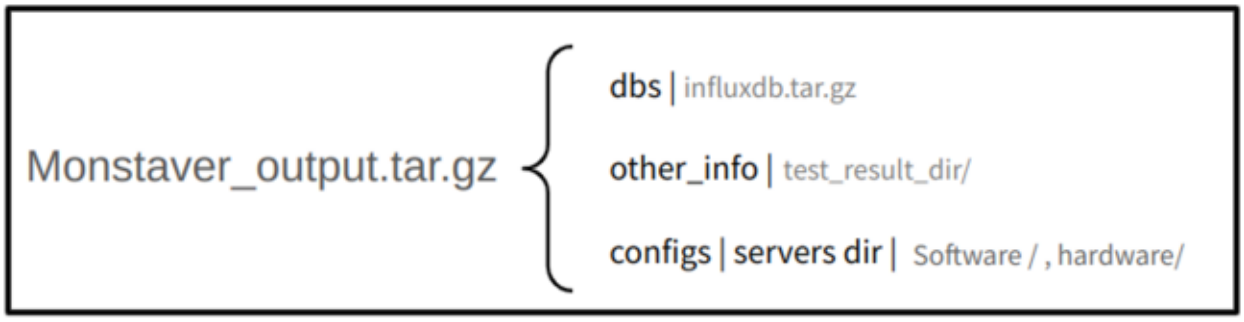
در این قسمت اطلاعات سرور mc دارای influxdb نوشته می‌شود. بیشتر کانفیگ ها مشابه بخش قبلی است فقط در قسمت databases از داخل فایل فشرده بکاپ اسم دیتابس مبدا استخراج می‌شود و مقدار prefix به آن اضافه شده و نام دیتابیس جدید که بکاپ روی آن بازگردانی خواهد شد ساخته می‌شود.

نکته: دیتابیس مبدا که در فایل فشرده قرار دارد باید حتما در influxdb وجود داشته باشد تا بتوان از روی آن داده ها بازگردانی شوند به دیتابیس جدید. این مورد از پیش نیاز های خود influx است و مربوط به معماری کارا نیست.

```
influxdbs_restore:

   MC:
      ip: 0.0.0.0
      ssh_port: 22
      ssh_user: user
      container_name: influxdb
      influx_volume: /var/lib/influxdb/KARA_RESTORE    # mount point inside
influxdb container
      databases:
         - prefix: "rst1_" # prefix of new database name
           location: /tmp/influxdb-backup/path/to/dbs/influxdb.tar.gz #
backup file location
```

۲-۳-۵-۲- خروجی برنامه:

فایل خروجی برنامه به صورت پیش فرض در مسیر /tmp/influxdb-backup/ قرار داشته و درون آن سه دایرکتوری اصلی وجود دارند: dbs برای بکاپ دیتابیس ها و other_info برای مسیر ها و فایل های دلخواه کاربر و configs برای اطلاعات سخت افزاری و نرم افزاری و swift که به ازای هر سرور هیولا وجود دارد.



شکل ۱۹: خروجی monstaver

۲-۳-۵-۳- روش استفاده از ابزار:

موارد ذکر شده در آرگومان های زیر در فایل کانفیگ نیز وجود دارند در صورت استفاده نکردن از آرگومان از فایل خوانده می‌شوند.

آرگومان های ورودی:

-t بازه زمانی بکاپ 'start time,end time'

-i مسیر یا فایل دلخواه کاربر برای قرار گرفتن در بکاپ

-sw بکاپ گرفتن از تنظیمات نرم افزاری سرور های هیولا

-hw بکاپ گرفتن از تنظیمات سخت افزاری سرور های هیولا

-s بکاپ گرفتن از کانفیگ های swift وring

-ib بکاپ گرفتن از دیتابیس

-d پاک کردن دایرکتوری خروجی نهایی ابزار و فقط ذخیره کردن خروجی در فرمتtar.gz

-r استفاده از ویژگیrestore

```
Python3 monstaver.py
```

۲-۳-۵-۴- توسعه ابزار: این ابزار از ۴ تابع اصلی تشکیل شده است که وظایف بکاپگیری و انتقال داده ها را دارند.

۱ـ backup: این تابع عملیات بکاپ را انجام می‌دهد و دو تابع backup_data_collector و info_collector را برای اجرای سریعتر در حالت multi threading اجرا می‌کند که در ادامه به شرح آنها پرداخته می‌شود. ورودی های این تابع شامل آرگومان هایی برای عملیات بکاپ از دیتابیس و مشخصات سرور و بازه زمانی و داده های موجود در فایل کانفیگ آن است.

```
def backup(time_range, inputs, delete, data_loaded, hardware_info,
software_info, swift_info, influx_backup):

# load data from yaml conf
# use convert time functions
if influx_backup:
       for mc_server, config in data_loaded.get('db_sources',
{}).items():
       for db_name in database_names:
              # make backup from influxdb

       # run backup_data_collector in multi threads for collect and
move data
# run info_collector in multi threads for collect swift and hardware
and software data

# upload backup to monster

backup_to_report = f"{backup_dir}/{time_dir_name}/"
return backup_to_report
```

در آخر مسیر دایرکتوری نهایی بکاپ به عنوان خروجی تابع قرار می‌گیرد.

۲ـ backup_data_collector: این تابع وظیفه دریافت و جابجایی و دسته بندی فایل های tar گرفته شده در هنگام بکاپ را دارد. در ورودی اطلاعات influx و ssh و container را دریافت می‌کند.

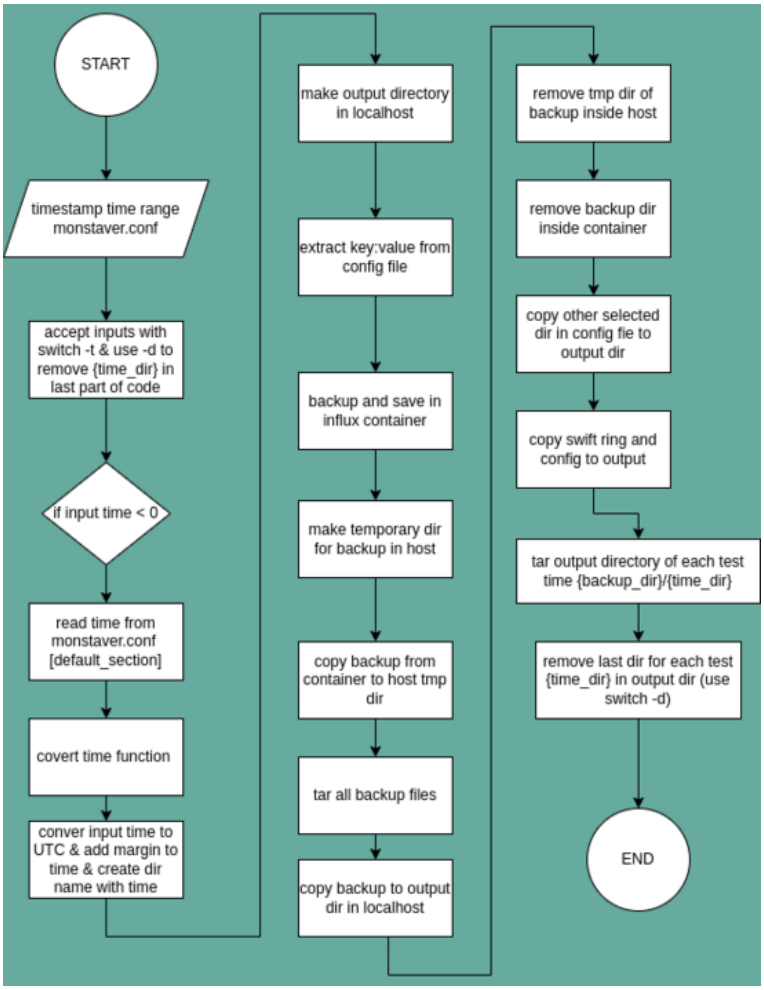
```
def backup_data_collector(ssh_port, ssh_user, ip_influxdb,
container_name, influx_volume, time_dir_name, bar, backup_dir):

# New_location_backup_in_host =
value['temporary_location_backup_host']
    tmp_backup = "/tmp/influxdb-backup-tmp"

# copy backup to temporary dir
# tar all backup
# move tar file to dbs dir inside your server (kara)
# remove temporary location of backup in host
# delete {time_dir} inside container
```

۳ـ info_collector: این تابع فرآیند های بکاپ گیری از اطلاعات سخت افزاری و نرم افزاری و swift هیولا را دارد و آنها را در دایرکتوری configs و دایرکتوری مختص هر سرور قرار می‌دهد.

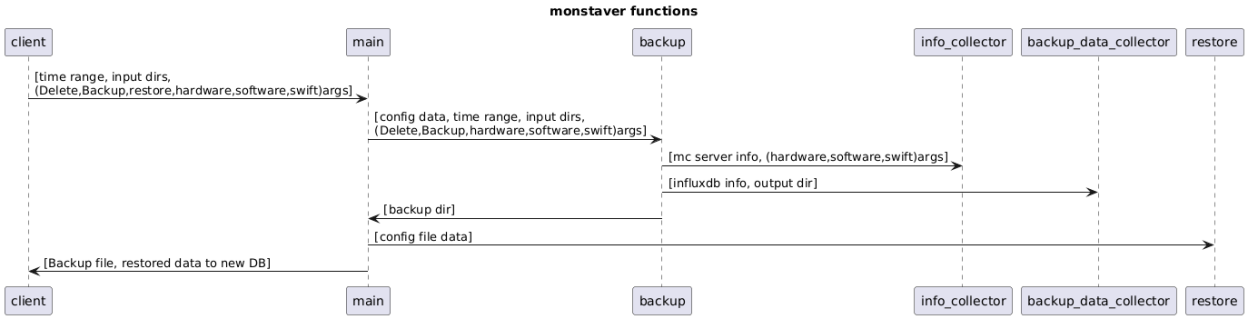
```
def info_collector(port, user, ip, backup_dir, time_dir_name,
container_name, bar, swift_info, hardware_info, software_info):

# make hardware/os/swift sub directories
# get swift config files and monster services
# extract docker compose file path and copy it
# copy etc dir from container to host
# copy container etc dir from host to your server
# copy host etc dir from host to your server
#### Execute commands to gather hardware information ####
#### Execute commands to gather OS information ####
```

۴ـ restore: این تابع وظیفه بازگردانی فایل فشرده شده بکاپ را به یک دیتابیس جدید دارد و به گونه ای که یک دیتابیس جدید با اسم مشابه دیتابیس مبدا بکاپ به علاوه یک مقدار prefix ایجاد می‌کند و قبل از انجام این عمل داده ها در یک دیتابیس موقت نیز بازگردانی می‌شوند. مراحل ذکر شده جزء مراحل استاندارد influxdb هستند. این تابع در ورودی خود داده های لود شده از فایل کانفیگ را دریافت می‌کند.

```
def restore(data_loaded):

for mc_server, config in data_loaded.get('influxdbs_restore',
{}).items():

       for db_info in databases:
              # Append the prefix to the extracted database name
              # Drop second_db
              # Create second_db
              # Ensure the target restore directory exists
              # Copy backup file to container mount point
              # Extract the backup.tar.gz
              # Restore backup to temporay database
              # Merge phase
              # Drop tmp db
              # remove untar and tar file in container
```



شکل ۲۰: فلوچارت *monstaver*



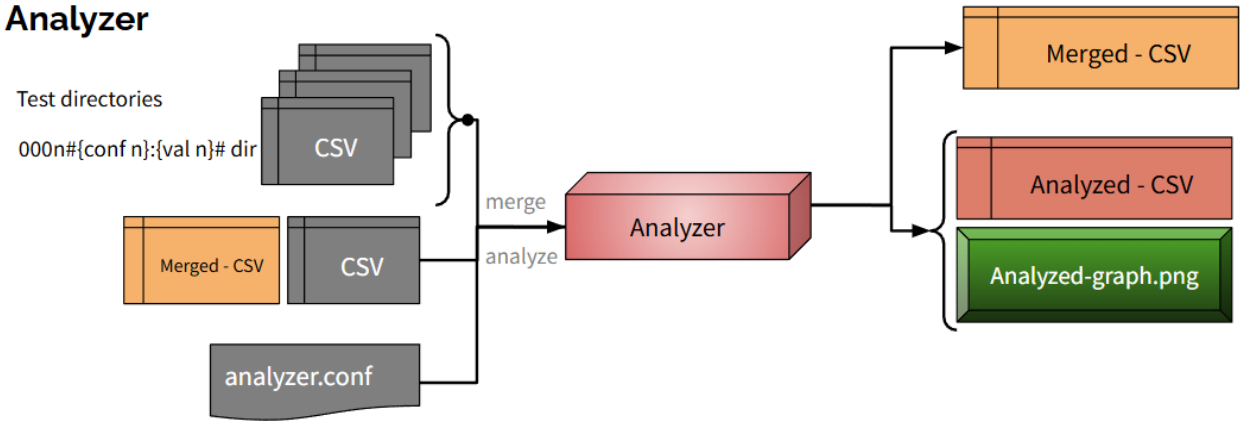
شکل ۲۱: توابع *monstaver*

۲-۳-۶- analyzer:

این ابزار از در مجموع وظیفه انجام دو فرآیند را دارد.

تحلیل یا آنالیز نتایج گزارش های دریافت شده از هیولا به صورت csv و ساخت تصاویر و گراف هایی آنالیز شده از آنها.

تجمیع نتایج گزارش ها به صورت یک فایل csv جامع برای همه گزارش های دریافت شده از هیولا در بازه های زمانی مختلف.



شکل ۲۲: روند کار *analyzer*

تجمیع (merge) :

ابزار برای فرآیند تجمیع نیاز به بیشتر از دو عدد فایل csv در ورودی دارد، می‌تواند آن ها را به صورت لیستی از فایل ها ("csv1,csv2,csv3,...") و یا دایرکتوری شامل چندین فایل ("*/csv_dir/") دریافت و پس از تجمیع آن ها نتیجه را در فایلی به نام merged.csv در مسیر داده شده توسط کاربر ذخیره کند.

تحلیل (analyze):

این بخش وظیفه تجزیه و تحلیل یک csv را بر عهده دارد، برای انجام این کار نیاز به تنظیم فایل کانفیگ analyzer.conf در مسیر /etc/kara/است.

از جمله عملیات های ریاضی که برای اعمال روی ستون های csv در این ابزار تعریف شده، می‌توان به میانگین(avg) ، جمع (sum)، ضرب (mul) و تقسیم (div) بین ستون های مورد نظر اشاره کرد. این ابزار می‌تواند برای تمام سطر های فایل ورودی دو مقدار میانگین و تجمیع ذخیره کند و در آخر یک csv جدید با ستون هایی جدید که نتایج تحلیل در آنها وجود دارد ایجاد کند.

روند کار بخش تحلیل به این صورت است که در ورودی یک فایل csv دریافت کرده و پس از آن از درون فایل کانفیگ، ابزار موارد مورد نیاز کاربر را دریافت و بر روی فایل اعمال می‌کند. در انتها خروجی این برنامه یک csv جدید است که نام آن از ترکیب اسم فایل اولیه و کلمه analyzed بوده است. برای درک بهتر کار کرد ابزار به بخش کانفیگ مراجعه کنید.

۲-۳-۶-۱- شرح فایل کانفیگ و اجزای هر بخش:

```
transformation:

    csv:
        columns:
            cpu.io: # new_column_name
                operation: avg   # values = avg - sum - mul - div
                selected_columns:
                    - mean_system.cpu.idle
                    - mean_system.cpu.iowait

            cpu.info: # new_column_name
                operation: sum   # values = avg - sum - mul - div
                selected_columns:
                    - sum_system.cpu.system
                    - sum_system.cpu.user

            total_cpu_Usage:
                operation: mul   # values = avg - sum - mul - div
                selected_columns:    # values = name of columns or a number
# just select two columns
                    - sum_system.cpu.system
                    - 96

            perUser_cpu_Usage:
                operation: div   # values = avg - sum - mul - div
                selected_columns:    # just select two columns
                    - sum_system.cpu.user
                    - sum_system.cpu.system
        rows:
            - sum
            - avg
```

۱-csv: در این قسمت بخش های مربوط به تحلیل و ساخت csv وجود دارد.

- columns: این قسمت نمایانگر تحلیل بر روی ستون های فایل ورودی است و عملیات های ریاضی مورد نیاز و ستون های انتخابی و نام ستون های جدید در زیر مجموعه این بخش قرار دارند. (در صورتی که نیاز به تحلیل بر روی ستون ها ندارید می‌توان کل این قسمت را حذف یا کامنت کرد)
    - در قسمت بعدی برای هر دسته از عملیات های تحلیل یک نام دلخواه در نظر گرفته شده است که برای اسم ستون جدید نیز در نظر گرفته می‌شود به عنوان مثال(cpu.io)
        - operation: عملیات های ریاضی موجود در این بخش نوشته می‌شوند.
        - selected_columns: در این قسمت لیستی از ستون های انتخابی برای تحلیل نوشته می‌شود و عملیات ریاضی روی آنها اعمال می‌شود برای عملیات تقسیم حتما

باید دو ستون انتخاب شوند و برای عملیات ضرب حتما دو ستون یا بیشتر برای
موارد دیگر در صورتی که کم تر از دو مورد باشند مقدار همان ستون چاپ می‌شود.

نکته: برای عملیات ضرب در این بخش می‌توان از اعداد نیز استفاده کرد تا در ستون انتخابی ضرب شوند.

- rows: در این بخش تحلیل میانگین و تجمیع برای همه ستون ها انجام شده و در دو سطر جدید با نام های AVG , SUM ذخیره می‌شوند.

بخش ساخت گراف تحلیل شده در فایل کانفیگ مشابه مثال ذکر شده در پایین بوده و در ادامه به شرح آن می‌پردازیم.

```
    graph:
        g1:    # group name
            filter:
                Host_name: # name of target column
                    - r1z1s1-controller # name of target value
                    - r2z2s2-controller
            selected_columns:
                - sum_system.cpu.user: sum_system.cpu.system   # X_axis:
Y_axis
                - mean_system.cpu.idle: mean_system.cpu.iowait
        g2:
            filter:
                Host_name:
                    - r2z2s2-controller
            selected_columns:
                - mean_system.cpu.idle: mean_system.cpu.iowait
```

۲- graph: در این قسمت بخش های مربوط به ساخت گراف بر حسب ستون های فایل csv ورودی قرار دارد.

- در قسمت اول این بخش یک نام دلخواه برای هر گروه از گراف ها قرار دارد که این نام در اسم تصویر نهایی نیز وجود دارد.
  - filter: در این بخش یک فرآیند فیلتر کردن وجود دارد برای جلوگیری از تداخل اسم ستون ها و مقداری آنها در زمان ساخت گراف. این بخش به این صورت کار می‌کند که در خط بعدی آن باید اسم ستونی که تفکیک کننده بخش مورد نیاز ما از دیگر بخش ها است نوشته شوند به عنوان مثال: ما نیاز به گرافی برای متریک های دو سرور r1z1s1 و r2z2s2 داریم در این بخش نام سرورها در ستون Host_name نوشته شده پس از این ستون برای تفکیک موارید مورد نیاز استفاده می‌کنیم.
  - selected_columns: در این بخش لیستی از ستون ها نوشته می‌شود که نیاز است گراف برای آنها ساخته شود. در هر خط از این لیست اسم دو ستون مقابل یک دیگر نوشته می‌شود با علامت ":" میان آنها ستون اول می‌شود محور x و ستون دوم محور y و برای آنها بر حسب یکدیگر گرافی ساخته می‌شود.

۲-۳-۶-۲- روش استفاده از ابزار:

آرگومان های ورودی:

-M نمایانگر عملیات merge یا تجمیع

-o مسیر خروجی برنامه

-sc لیستی از csv های مورد نیاز برای تجمیع (csv1,csv2,csv3,...) یا دایرکتوری شامل چندین فایل(/dir/*)

-A نمایانگر عملیات analyze یا تحلیل

-c اسم و مسیر فایل csv نیازمند تحلیل

-k نمایانگر اینکه ستون های اولیه فایل تحلیل شده در آن باقی بمانند یا حذف شوند و فقط ستون های جدید در آن باشند.

-G نمایانگر عملیات ساخت گراف

```
python3 analyzer.py -M -o <output> -sc <csv1,csv2,csvn> -A -c <csv for
analyze> -k -G
python3 analyzer.py -M -o <output> -sc <csv_dir/*> -A -c <csv for analyze> -k
-G
```

۲-۳-۶-۳- توسعه ابزار: این ابزار از دو بخش تشکیل شده است بخش اول توابعی که اسناد موجود در کاتب را تحلیل می‌کند این توابع توسط ابزار report_recorder فراخوانی می‌شوند. بخش دوم توابع اصلی و مورد استفاده مستقیم در خود ابزار analyzer که در مجموع از ۳ تابع اصلی ساخته شده است که در ادامه شرح داده می‌شوند.

۱ـ بخش اول توابع تحلیل اسناد:

این قسمت از دو دسته تابع تشکیل شده است دسته اول توابعی که اسناد و اطلاعات سخت افزار را تحلیل می‌کنند دسته دوم توابعی که اسناد و اطلاعات نرم افزاری را تحلیل می‌کنند.

- دسته اول (سخت افزار) : این توابع در ورودی خود نیاز به فایل های اطلاعات سخت افزاری به دست آمده از monstaver و بکاپ آن را دارند. این دسته شامل ۸ تابع است که در شبه کد پایین شرح داده شده اند.

```
def compare(part, spec):
       # compare between servers hardware info
       return dict

def generate_model(server, part, spec):
       # manage and separate each hardware file process by which
function

def generate_disk_model(serverName):
       # lshw -C disk

def generate_motherboard_model(serverName):
       # dmidecode -t 2

def generate_net_model(serverName):
       # lshw -json -C net

def generate_memory_model(serverName):
       # lshw -short -C memory

def generate_cpu_model(serverName):
       # lscpu

def generate_brand_model(serverName):
       # dmidecode -t 1
```

- دسته دوم(نرم افزار): : این توابع در ورودی خود نیاز به فایل های اطلاعات نرم افزاری به دست آمده از monstaver و بکاپ آن را دارند. این دسته شامل ۱۰ تابع است که در شبه کد پایین شرح داده شده اند.

```
def generate_confs (confType, serverType = None):
    # read and compare configs of servers
    return compared_dict

def get_conf (server, confType, serverType = None):
    # read hardware config files

def partitioning (confOfServers , confType , unimportantconfDir):
    # separate important and unimportant conf inside documents
    return confOfServers

def extract_ini_file (server, serverType):
    # read conf file in "ini" format

def generate_ring (servername, serverType):
    # make ring data and compare them

def generate_all_swift_status(services):
    # make swift configs data and compare them
def generate_swift_status(servername):
    # make swift service status data

def get_uncommonConf (conf1, conf2, confType):
    # compare and separate uncommon configs

def get_commonConf (conf1, conf2, confType):
    # compare and separate common configs

def compare_confs (confsOfServers, confType):
    # compare all configs
```

در آخر تابع compare در سخت افزار و دو تابع generate_confs و partitioning در نرم افزار ذکر شده دیکشنری هایی را در خروجی خود دارند که در ابزار report_recorder تبدیل به جداولی در html می‌شوند.

۲ـ merge_csv: این تابع اصلی ترین بخش عملیات تجمیع را انجام می‌دهد. در ورودی خود یک فایل csv و مسیر خروجی و دو دیکشنری برای فایل های merge و merge_info دریافت می‌کند. در صورتی که این تابع در manager استفاده شود دو دیکشنری ذکر شده را دریافت و آنها را می‌سازد و اگر ابزار به صورت مستقیم مورد استفاده قرار گیرد فقط فایل تجمیع شده را تحویل می‌دهد.

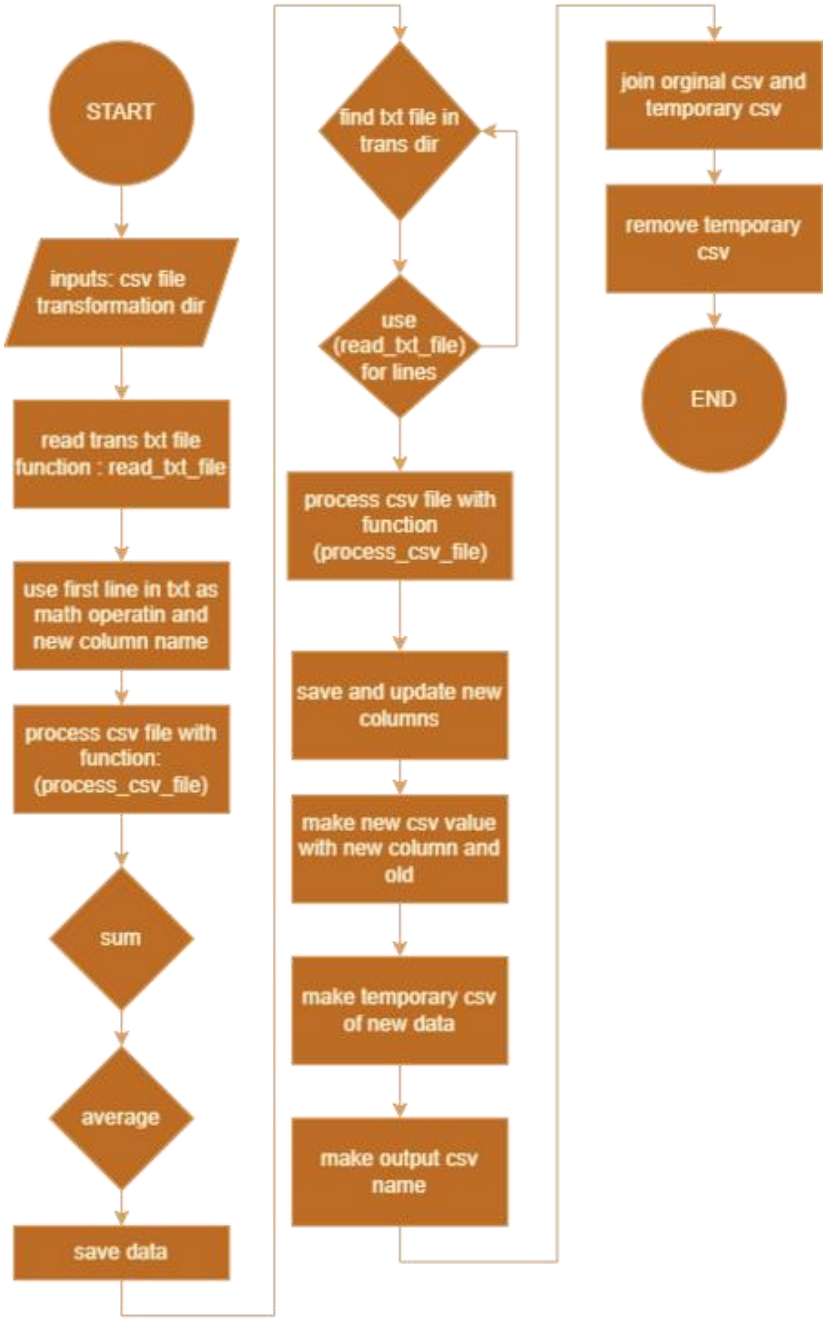
```
def merge_csv(csv_file, output_directory, mergedinfo_dict,
merged_dict):

if mergedinfo_dict:
    # make merged_info file

if merged_dict:
    # make merged file

# make csv
merged.to_csv(f'{output_directory}/merged.csv', index=False,
mode='w')

return f'{output_directory}/merged.csv'
```

۳ـ analyze_and_save_csv: این تابع وظیفه اصلی تحلیل csv ورودی را دارد و پس از دریافت کانفیگ های مورد نیاز از فایل، عملیات های تحلیل و ساخت سطر و ستون های جدید را در فایلی جدید انجام می‌دهد و در صورت نیاز می‌توان با آرگومانی مشخص کرد که ستون های قدیمی و اولیه در فایل جدید وجود داشته باشند یا خیر.

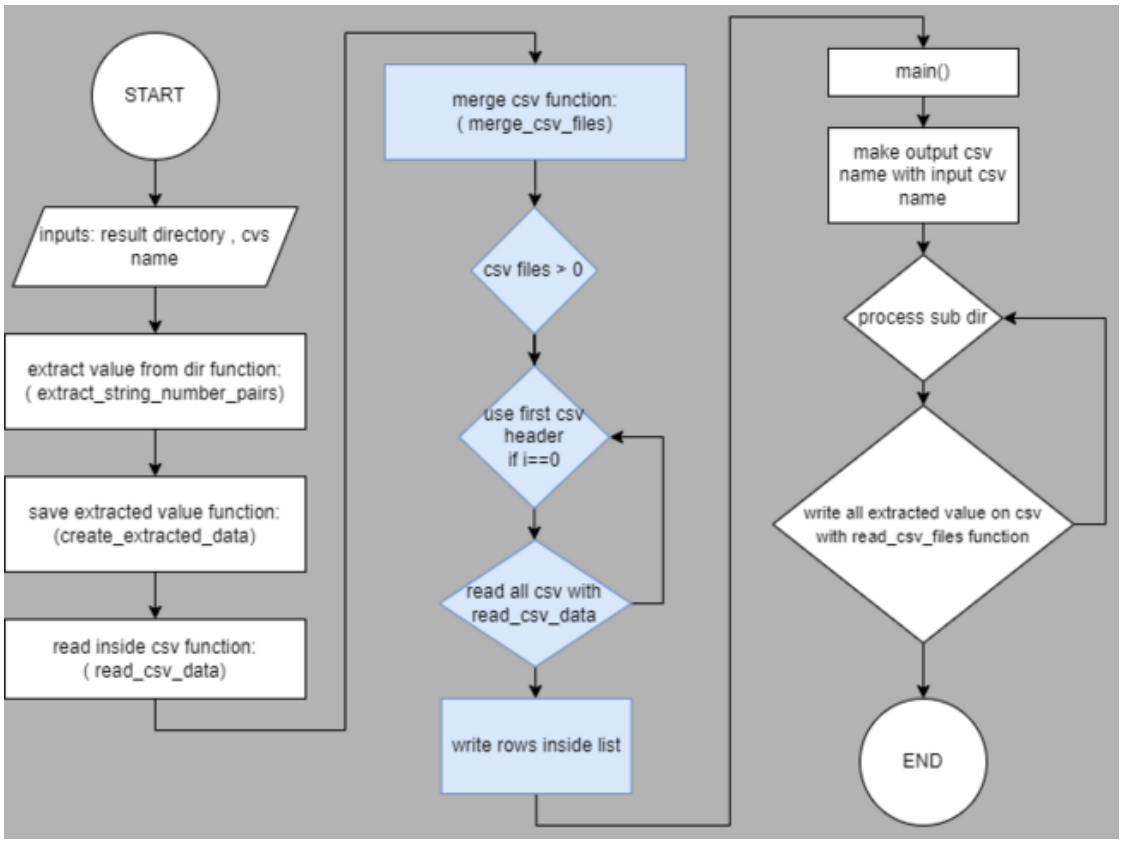
```
def analyze_and_save_csv(csv_original, keep_column, output_directory,
data_loaded):

for section_name, content in data_loaded['transformation'].items():

       if 'csv' in section_name:
              # user need csv to analyze and make it

              if 'columns' in content:
                     for new_column_name, column_content in
content.get('columns', {}).items():
                     # analyze and make new columns

              if row_operations:
                     # analyze and make rows

return final_output_csv_path
```

۴ـ plot_and_save_graph: این تابع وظیفه ساخت گراف را از csv دارد به این صورت که دو ستون را می‌تواند نسبت به هم در محور های y , x قرار دهد. در فایل کانفیگ نیز مقادیر مورد نیاز این تابع قرار داده شده اند.

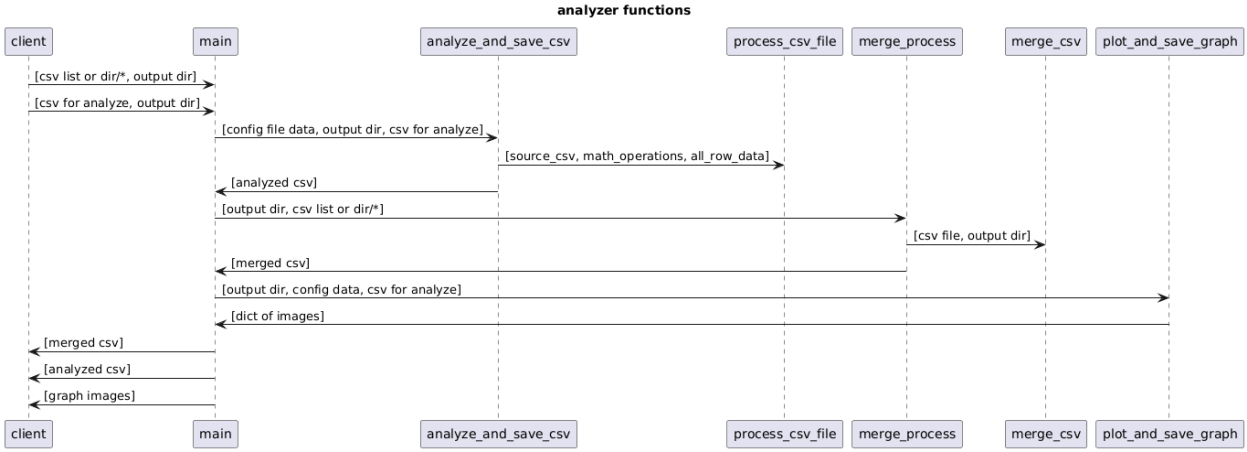
```
def plot_and_save_graph(csv_original, output_directory, data_loaded):

for group_name, group_data in
data_loaded['transformation']['graph'].items():

       for filter_column, filter_values in filter_data.items():
              # make temp csv

       for column_pair in selected_columns:
              for x_column, y_column in column_pair.items():

                if x_column in temp_csv.columns and y_column in
temp_csv.columns:
                     # plot graph and make image

return image_dict
```



شکل ۲۳: فلوچارت بخش *analyze*



شکل ۲۴: فلوچارت بخش *merge*



شکل ۲۵: توابع *analyzer*

۲-۳-۷- report_recorder:

قبل از اجرای این ابزار چه در سناریو manager و چه به صورت مجزا نیاز است که فایل کانفیگ pywikibot که ابزار رابط بین report_recorder و کاتب است کانفیگ شود. برای این کار به مسیر کتابخانه های پایتون

/usr/local/lib/python <n> /dist-packages/report_recorder_bot/pywikibot/families/

مراجعه کنید این مسیر می‌تواند بسته به ورژن پایتون متفاوت باشد. پس از آن فایل kateb_family.py را ویرایش کرده و دامنه و نام کاربری و و رمز عبور وب سرور کاتب را در این بخش ذکر کنید مشابه مورد زیر:

نکته: درصورتی که وب سرور کاتب فاقد authentication است در این بخش فقط دامنه کاتب ذکر شود.

```
    name = 'kateb'
    langs = {
        'fa': 'user:pass@kateb.burna.ir',  # uesr:pass of web
server@kateb_domain  Attention!! do not type '#' in user or pass here,
replace it with Ascii code "%23"
    }
```

نکته: در صورتی که در نام کاربری یا رمز وب سرور از علامت "#" استفاده شده بود کد جدول Ascii آن یعنی 23 را به فرمت 23% بنویسید.

پس از انجام فرآیند ذکر شده نیاز است فایل user-config.py در دایرکتوری خود ابزار یا manager بسته به نیاز و اجرای ابزار ویرایش شده و فقط نام کاربری سامانه کاتب در آن قرار گیرد. فقط در اولین اجرا password نیز از کاربر سوال شده و در یک فایل رمزنگاری شده ذخیره سازی می‌شود.

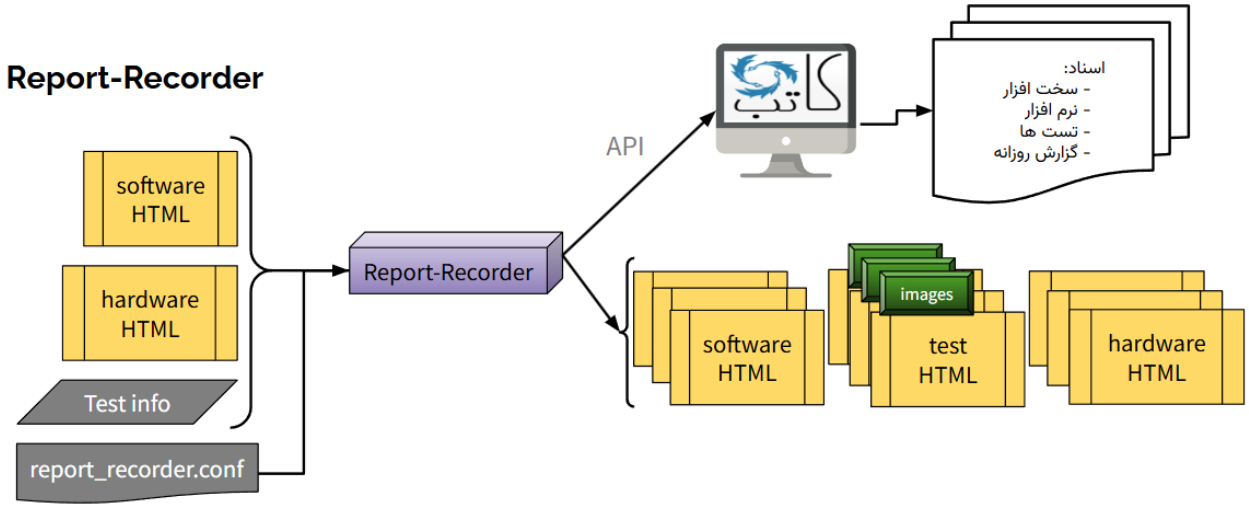
```
mylang = 'fa'
family = 'kateb'
usernames['kateb']['fa'] = 'user name'
console_encoding = 'utf-8'
```

۲-۳-۷-۱- روند کار ابزار:

این ابزار وظیفه ایجاد مستندات از گزارش های دریافت شده برای تست های هیولا را دارد، آنها را در فرمت html ذخیره و سپس در سامانه کاتب بارگذاری می‌کند.

روند کار به این صورت است که در ورودی یک قالب html برای مشخصات سخت افزاری سرور های هیولا و یک قالب دیگر برای مشخصات نرم افزاری هیولا دریافت کرده و درون آنها در مکان هایی مشخص و با استفاده از placeholder هایی مشخص اطلاعات سرور ها را جایگذاری می‌کند و چندین نوع قالب از آنها در مسیر خروجی می‌سازد.

برای اطلاعات تست های هیولا نیز فایل csv تجمیع شده و دایرکتوری خروجی تست ها را دریافت و یکسری مستندات html برای مشخصات و کانفیگ های تست ساخته و سپس آنها همراه با گراف های هر تست در کاتب آپلود می‌کند.



شکل ۲۶: روند کار report-recorder

۱- قالب های html اولیه سخت افزار و نرم افزار:

این دو فایل در دایرکتوری input_templates وجود دارند، مشابه قالب های ذکر شده در زیر از placeholder های {sw_config} برای درج مشخصات نرم افزاری و {hw_config} برای درج مشخصات سخت افزاری استفاده می‌شود.

موارد نوشته شده مقابل این placeholder ها المان هایی هستند که نمایانگر کانفیگ یا اطلاعاتی مورد نیاز برای چاپ در این بخش از سند هستند.

پیشنهاد می‌شود از همین موارد پیشفرض استفاده شود و یا در صورت نیاز المان ها حذف یا جابجا شوند و یا متن های موجود در فایل ها تغییر کنند ولی مورد جدیدی به جز موارد نوشته شده اضافه نگردد.

```
<h2>مشخصات سخت افزاری</h2>
<p> در این بخش مشخصات سخت افزاری هر کدام از سرور ها به تفصیل مورد بررسی قرار
می گیرد. این مشخصات بر اساس قسمت های مختلف سخت افزار سرورها تقسیم بندی شده
اند.</p>

<h3> سرور </h3>
{hw_config}:hardware,brand

<h3>پردازنده</h3>
{hw_config}:hardware,cpu
<a href="./subpages/{title}--CPU"> مشخصات سخت افزار پردازنده </a>

<h3>حافظه اصلی</h3>
{hw_config}:hardware,memory
<a href="./subpages/{title}--Memory"> مشخصات سخت افزار حافظه </a>
```

```
<h3>شبکه</h3>
{hw_config}:hardware,net
<a href="./subpages/{title}--Network"> مشخصات سخت افزار شبکه </a>

<h3>دیسک و کنترولر</h3>
{hw_config}:hardware,disk
<a href="./subpages/{title}--Disk"> مشخصات سخت افزار دیسک و کنترولر </a>

<h3>PCI</h3>
<p> برای مشاهده اطلاعات به سند <a href="./subpages/{title}--PCI">  اطلاعات pci
</a> مراجعه کنید.</p>

<h3>مادربورد</h3>
{hw_config}:hardware,motherboard

<h3>تنظیمات RAID</h3>
<p>این مشخصات بسته به برند سرور فیزیکی با دستورات متفاوتی گزارش می شود.</p>
<p>استفاده می شود ssacli از دستور HP به طور مثال برای سرور های.</p>
<p>در این سند مشخصات نرم افزاری کلاستر آمده است</p>

<h2>معماری کلاستر</h2>

<h3>نسخه نرم افزارهای استفاده شده در هر سرور</h3>
{sw_config}:software_version

<h2>مشخصات نرم افزاری</h2>
<p> در این بخش مشخصات نرم افزاری هر کدام از سرورها مورد بررسی قرار می گیرد.
این مشخصات به دو بخش کلی تنظیمات مربوط به هیولا و تنظیمات مربوط به سیستم تقسیم
بندی می شود.</p>

<h3>تنظیمات مربوط به هیولا</h3>
<p> این تنظیمات شامل سرویس ها، وضعیت رینگ ها و فایل های کانفیگ هر سرویس می
شود.</p>

<h4>وضعیت سرویس های هیولا</h4>
<h5>سرویس های اصلی</h5>
{sw_config}:swift_status,main

<h5>سرویس های آبجکت</h5>
{sw_config}:swift_status,object

<h5>سرویس های اکانت</h5>
{sw_config}:swift_status,account

<h5>سرویس های کانتینر</h5>
{sw_config}:swift_status,container

<h4>وضعیت رینگ ها</h4>
<h5>رینگ های آبجکت</h5>
{sw_config}:rings,object

<h5>رینگ های اکانت</h5>
{sw_config}:rings,account

<h5>رینگ های کانتینر</h5>
{sw_config}:rings,container
```

```
<h4>فایل های کانفیگ سرورها</h4>
<h5>پروکسی سرور ها</h5>
{sw_config}:server_confs,proxy

<h5>آبجکت سرور ها</h5>
{sw_config}:server_confs,object

<h5>اکانت سرور ها</h5>
{sw_config}:server_confs,account

<h5>کانتینر سرور ها</h5>
{sw_config}:server_confs,container

<h3>تنظیمات مربوط به سیستم</h3>
<p> در این قسمت تنظیمات مربوط به کرنل و سرویس های داخل داکر به ازای هر سرور
قرار می گیرد.</p>

<h4>وضعیت سرویس ها</h4>
<h5>systemctl list-units</h5>
{sw_config}:systemctl

<h5>lsof | wc -l</h5>
{sw_config}:lsof

<h5>lsmod</h5>
{sw_config}:lsmod

<h4>تنظیمات sysctl</h4>
{sw_config}:sysctl
```

۲- تحلیل اسناد:

سند یا فایل html ساخته شده برای مشخصات سخت افزاری و نرم افزاری دارای تحلیل یا آنالیز بوده و تفاوت ها و نقاط مشترک سرور های هیولا چه از نظر سخت افزار و چه نرم افزار در آن ثبت شده است تا فرآیند عیب یابی سریع تر انجام شود. این تحلیل انجام شده برای مشخصات نرم افزاری مقداری پیشرفته تر بوده و می‌توان در زمان نمایش اشتراکات و تفاوت های سرور ها مواردی که مهم نیستند و اولویتی در تحلیل ندارند را با استفاده از فایل هایی که این موارد غیر ضروری در آنها ذکر شده اند فیلتر کرد.

روش کار این بخش به این صورت که در مسیر خروجی نهایی و ذخیره html ها یک دایرکتوری با نام unimportant_conf باید ایجاد کنید و درون آن فایل های برای هر بخش از سند که نیاز به تحلیل بیشتر دارد بسازید و داده های غیر ضروری را درون آنها قرار دهید مانند مثال ذیل:

ما در دایرکتوری و مسیر ذکر شده فایلی با نام sysctl-Unimportant_conf.txt ایجاد کرده این و درون آن می‌توان هر مقداری که اهمیتی در سند ندارد قرار دهیم مانند "kernel.random.boot_id" با این کار در بخش sysctl سند نرم افزار این مقدار قرار داد شده که از این دستور استخراج می‌شود در بخش غیر مهم جدول sysyctl در سند نوشته می‌شود و باقی موارد در بخش مهم جدول.

۲-۳-۷-۲- شرح فایل کانفیگ و اجزای هر بخش:

قبل از اجرای نرم افزار نیاز است که اطلاعات کاربری کاتب خود را در فایل user-config.py موجود در دایرکتوری-report recorder یا manager بسته به محل فراخوانی ابزار وارد نمایید. پس از باز کردن این فایل نام کاربری خود را در قسمت user nameقرار دهید و پس از اولین اجرای نرم افزار و به هنگام آپلود اطلاعات در کاتب فقط در دفعه اول password خود را وارد می‌کنید رمز شما در یک فایل md5 ذخیره می‌شود. فایل کانفیگ برنامه نیز در مسیر /etc/kara/و با اسم خود برنامه وجود دارد.

۱- بخش اول فایل کانفیگ شامل اطلاعات اولیه و اصلی به شرح ذیل بوده:

```
cluster_name: kara
scenario_name: test

tests_info:
  merged: /path/to/kara/results/analyzed/merged.csv
  merged_info: /path/to/kara/results/analyzed/merged_info.csv
  images_path: /path/to/results/
  test_tags:
    - "تست"
    - "کارایی"
    - "هیولا"

hw_sw_info:
  configs_dir: /tmp/influxdb-backup/path/to/backup dir
  software_template:
/path/to/kara/report_recorder/input_templates/software.html
  software_tags:
    - "گزارش‌ها"
    - "عامل سیستم"
    - "هیولا"
  hardware_template:
/path/to/kara/report_recorder/input_templates/hardware.html
  hardware_tags:
    - "گزارش‌ها"
    - "افزار سخت"
    - "هیولا"

kateb_list_page: "name of page" # the page include list of titles
output_path: /path/to/kara/report_recorder/output_htmls/
```

- cluster_name: اسم کلاستر هیولا که گزارش ها برای آن ایجاد می‌شوند.
- scenario_name: اسم سناریو یا نوع سندی که برای آن کلاستر ساخته خواهد شد.
- merged: فایل تجمیع شده همه تست های یک دسته.
- tests_info: بخش مربوط به اطلاعات اولیه و مورد نیاز برای اسناد تست های گرفته شده از هیولا.

- merged: فایل تجمیع شده تمام csv های یک دسته تست گرفته شده.
- merged_info: فایل تکمیلی اطلاعات هیولا در زمان هر تست به صورت تجمیع شده.
- **این دو فایل ذکر شده در بخش بالا هم زمان با هم ایجاد می‌شوند و مکمل یکدیگر هستند.**
- images_path: مسیر دایرکتوری والد همه تست ها که شامل تصاویر و گراف تست ها است.
- test_tags: رده های اسناد نتایج تست.

- hw_sw_info: بخش مربوط به اطلاعات اولیه و مورد نیاز برای اسناد سخت افزار و نرم افزار هیولا و سرور آن.
  - configs_dir: اطلاعات موجود در اسناد از یکی از بکاپ های گرفته شده توسط ابزار monstaver استخراج می‌شود برای این منظور باید آدرس دایرکتوری بکاپ که در حالت فشرده نیست را در این بخش وارد کرد.
  - software_template و hardware_template: نام فایل های html ورودی و آدرس آنها.
  - software_tags و hardware_template: رده های مورد استفاده در این نوع اسناد به تفکیک.

- kateb_list_page: اسم صفحه ای در کاتب که لیستی از نام سند های گزارش به آن اضافه می‌شود، این صفحه اگر وجود نداشت ساخته می‌شود اگر وجود داشت به انتهای آن لیست صفحات اضافه می‌شود.

- output_path: مسیر خروجی و ذخیره همه html های ساخته شده توسط ابزار که دارای دو دایرکتوری برای صفحات فرعی و تصاویر موجود در اسناد است.

۲- بخش دوم فایل کانفیگ شامل اطلاعات مورد نیاز برای سند تست های هیولا بوده و دسته بندی اسناد و شکستند سند به چندین زیر مجموعه در این قسمت مشخص می‌شود.

```
classification:
  categories:
    LAT:
      comment: "می‌شود بررسی کلاستر تاخیر میزان تستها این در."
      filter:
        workload.concurrency:
          - 1
      categories:
        P1:
          filter:
            workload.proxy:
              - 1
        P2:
          filter:
            workload.proxy:
              - 2
        P3:
          filter:
```

```
            workload.proxy:
              - 3
    BW:
      filter:
        workload.concurrency:
          - 72
          - 144
    TP:
      filter:
        workload.concurrency:
          - 256
          - 512

  maxTestsPerPage: 8 #Specifies the maximum number of tests that can be
displayed on a single web page when the 'auto-divider=True', default: 8
  autoDivider: False #default: True
  comment: "می‌شود داده نمایش سند ابتدای در متن این."
```

- classification: این بخش نحوه شکسته شدن اسناد و بخش‌بندی آن‌ها را بر اساس پارامترهای تست مشخص می‌کند که شامل موارد زیر است:
  - categories: نام بخش‌های مختلف و نحوه فیلتر کردن آن‌ها در این بخش مشخص می‌شود. در مثال بالا بخش تمامی تست‌ها به سه دسته اصلی LAT-BW-TP تقسیم می‌شوند و دسته LAT نیز خود به سه بخش P1-P2-P3 تقسیم شده است.
    - comment: هر بخش می‌تواند یک کامنت داشته باشد. کامنت متنی است که در ابتدای آن دسته نمایش داده می‌شود.
    - filter: در اینجا فیلترهای مربوط به هر دسته مشخص می‌شود. در مثال بالا تمامی تست‌ها با workload.concurrency=1 در بخش LAT و از بین این تست‌ها، آن‌هایی که مقدار workload.proxy آن‌ها برابر با ۱ است در زیر بخش P1 قرار می‌گیرند. همچنین تست‌هایی که در آن‌ها مقدار workload.concurrency برابر با ۷۲ یا ۱۴۴ است در بخش BW قرار می‌گیرند.
  - maxTestsPerPage: حداکثر تعداد تست یا گزارشی که در یک صفحه html یا کاتب قرار می‌گیرد.
  - autoDivider: در صورتی که مقدار این پارامتر False باشد، شکستن اسناد تنها توسط دسته‌بندی‌ها و فیلترهای مشخص شده انجام می‌شود و مقدار maxTestsPerPage در نظر گرفته نمی‌شود.
  - comment: متن قرار گرفته شده در این قسمت در ابتدای سند نمایش داده می‌شود.

۲-۳-۷-۳- روش استفاده از ابزار:

در صورتی که فایل کانفیگ تنظیم شده باشد نیازی به استفاده از اکثر آرگومان های ذکر شده نیست و فقط با دستور ذکر شده در انتها می‌توان ابزار را اجرا کرد.

آرگومان های ورودی:

-st مسیر و فایل قالب سند نرم افزار

-ht مسیر و فایل قالب سند سخت افزار

-tp عملیات ساخت فایل های html برای تست های گرفته شده

-o مسیر خروجی تمام فایل های ساخته شده

-cn اسم کلاستر

-sn اسم سناریو

-cd مسیر دایرکتوری بکاپ گرفته شده در ابزارmonstaver

-m فایل csv تجمیع شده(merged)

-mi فایل تکمیلی csv تجمیع شده(merged_info)

-kl اسم یک سند کاتب برای ایجاد لیست صفحات

-img مسیر دایرکتوری والد همه تست ها و تصاویر

-H عملیات ساخت html فقط ساخت و ذخیره فایل ها به صورت لوکال

-U عملیات آپلود در کاتب (اگر در کنار H- استفاده شود همین سند های جدید آپلود می‌شوند اگر مجزا استفاده شود در مسیر خروجی اگر فایلی از قبل بود آپلود می‌شود)

```
python3 report_recorder.py -H -U
```

۲-۳-۷-۴- توسعه ابزار:

این ابزار از یک کلاس و ۱۰ تابع اصلی و یک بات تشکیل شده است که در ادامه به شرح هر یک پرداخته شده است.

۱_ pywikibot:

این ابزار از نرم افزار pywikibot برای ارسال مستندات به پلتفرم mediawiki (کاتب) استفاده می‌کند و برای تحلیل اسناد از ابزار analyzer و برای ساخت صفحات سند در فرمت html از کتابخانه dominate بهره می‌برد.

به خاطر وجود سیستم کپچا در سامانه کاتب نیاز به تغییراتی در کد نرم افزار pywikibot بود تا بتواند از طرق API با سرور در ارتباط باشد و سوال های ریاضی موجود را پاسخ دهد تا اسناد آپلود شوند. برای این کار کد زیر در فایل apisite.py_ در مسیر pywikibot/site/ تغییر پیدا کرده.

کلاس APISite تابع editpage خط 2141

```
if captcha['type'] in ['math', 'simple']:
                            if "+" in captcha['question']:
                                resultCaptcha =
str(int(captcha['question'].split("+")[0])+int(captcha['question'].spl
it("+")[1]))
                            else:
                                resultCaptcha =
str(int(captcha['question'].split("-")[0])-
int(captcha['question'].split("-")[1]))
                            #req['captchaword'] =
input(captcha['question'])
                            req['captchaword'] = resultCaptcha
                            continue
```

۲_ create_sw_hw_htmls : این تابع وظیفه ساخت سند یا html های مربوط به گزارش های سخت افزاری و نرم افزاری را دارد و برای این کار نیاز به قالب های اولیه html با فرمت خاص و همچنین داده های درون فایل کانفیگ دارد.

```
def create_sw_hw_htmls(template_content, html_output, page_title,
data_loaded):

#HW_page_title = cluster_name
#SW_page_title = cluster_name + scenario_name

       if 'hw_config' in template_content:
              for hconfig_info in re.finditer(r'{hw_config}:(.+)',
template_content):

                     # analyze and compare hardware data and convert
dictionary to html
```

```
                    dict = analyzer.compare(part.strip(),
spec.strip())
                    hw_info_dict.update({spec.strip():dict})
                    html_of_dict = dict_html_hardware(dict)

                    # replace created html code with placeholder in
template

       if 'sw_config' in template_content:
              for sconfig_info in re.finditer(r'{sw_config}:(.+)',
template_content):
                    # analyze and compare software data and convert
dictionary to html


                    if sconfigs[0] == "swift_status":
                software_html = dict_html_software()
            else:
                # partitioning configs
                configs = analyzer.partitioning()
                software_html =
dict_html_software(configs,sconfigs[0])

htmls_dict.update({page_title:html_data})

if 'hw_config' in template_content:

        # make hardware sub pages
        htmls_dict.update(sub_pages_maker(html_data, page_title,
hw_info_dict, data_loaded))

return htmls_dict
```

در آخر این تابع دیکشنری از صفحات html و تیتر آنها ساخته می‌شود که در تابع convert_html_to_wiki به فرمت خاص ویکی تبدیل شده و سپس در سامانه آپلود می‌شوند.

۳_ create_test_htmls: در این تابع اسناد تست های گرفته شده از هیولا ساخته می‌شوند. فرآیند ساخت صفحات این بخش توسط کلاس testClassification و توابع آن انجام می‌شوند. در ورودی این تابع علاوه بر تیتر صفحات و مسیر خروجی نیاز به دو فایل merged_info , merged و دیکشنری تصاویر ساخته شده توسط status_reporter نیاز است، این دیکشنری می‌تواند توسط خود ابزار status_reporter ارسال شود و یا تابع path_to_dict از دایرکتوری والدی که همه تصاویر در آن قرار دارند این دیکشنری را بسازد.

```
def create_test_htmls(html_output, cluster_name, scenario_name,
merged_file, merged_info_file, imgsdict, yamlConf):

#page_title = cluster_name + scenario_name

move_images(imgsdict,os.path.join(html_output,'subpages/imgs'))

    tc = testClassification()

    for page in tc.AllPagesHTML:
       # save pages to html file

return tc.AllPagesHTML
```

۴ـ path_to_dict: این تابع دایرکتوری والد تصاویر (نتایج گزارش و تست) را دریافت و از دایرکتوری های زیر مجموعه و تصاویر آنها یک دیکشنری ایجاد می‌کند برای ارسال به تابع create_test_htmls تا تصاویر داشبورد از آنها استخراج شده و در سند قرار گیرند.

```
def path_to_dict(img_path_or_dict):
imgs_path_to_dict = {}
for time_dir in os.listdir(img_path_or_dict):
     if os.path.isdir(time_path):
          imgs_path_to_dict[time_dir] = {}
              for host_dir in os.listdir(time_path):
                  if "-images" in host_dir:
                      host_path = os.path.join(time_path, host_dir)
                         if os.path.isdir(host_path):
                            imgs_path_to_dict[time_dir][host_dir] = {}
                            # Collect all images in the host directory
                            # Separate dashboard images from others
       imgs_path_to_dict[time_dir][host_dir][dashboard_name_clean] =
dashboards
return imgs_path_to_dict
```

۵ـ convert_html_to_wiki: این تابع در ورودی مقداری داخل فایل های html را دریافت و آنها را به فرمت خاص mediawiki تبدیل می‌کند تا سند به درستی نمایش داده شود.

```
def convert_html_to_wiki(html_content):
# Convert dominate document to a string if needed
    if isinstance(html_content, document):
        html_content = html_content.render()
    elif not isinstance(html_content, str):
        html_content = str(html_content)
    try:
        soup = BeautifulSoup(html_content, 'html.parser')
       # Convert <a> tags to wiki links
       # Convert <img> tags to wiki images
       # Remove <body>, <thead>, and <tbody> tags
```

۶ـ upload_data: در این تابع صفحات با فرمت ویکی در کاتب آپلود می‌شوند و در آخر لیستی از صفحات آپلود شده در یک صفحه دلخواه کاربر قرار می‌گیرد. در ورودی این تابع اطلاعات login کاتب نیز قرار دارد.

```
def upload_images(site, html_content):
       for title, content in title_content_dict.items():
            page = pywikibot.Page(site, title)
            page.save(summary="Uploaded by KARA", force=True,
quiet=False, botflag=True)
       if kateb_list:
              # Check if the page exists
              # Get the current content of the page
              # Define the new page name to append
```

۷ـupload_images: این تابع تصاویر موجود در اسناد ساخته شده را آپلود می‌کند. ابتدا صفحات html که درون آنها آدرس تصاویر قرار دارد را خوانده و سپس تصویر را استخراج و آپلود می‌کند.

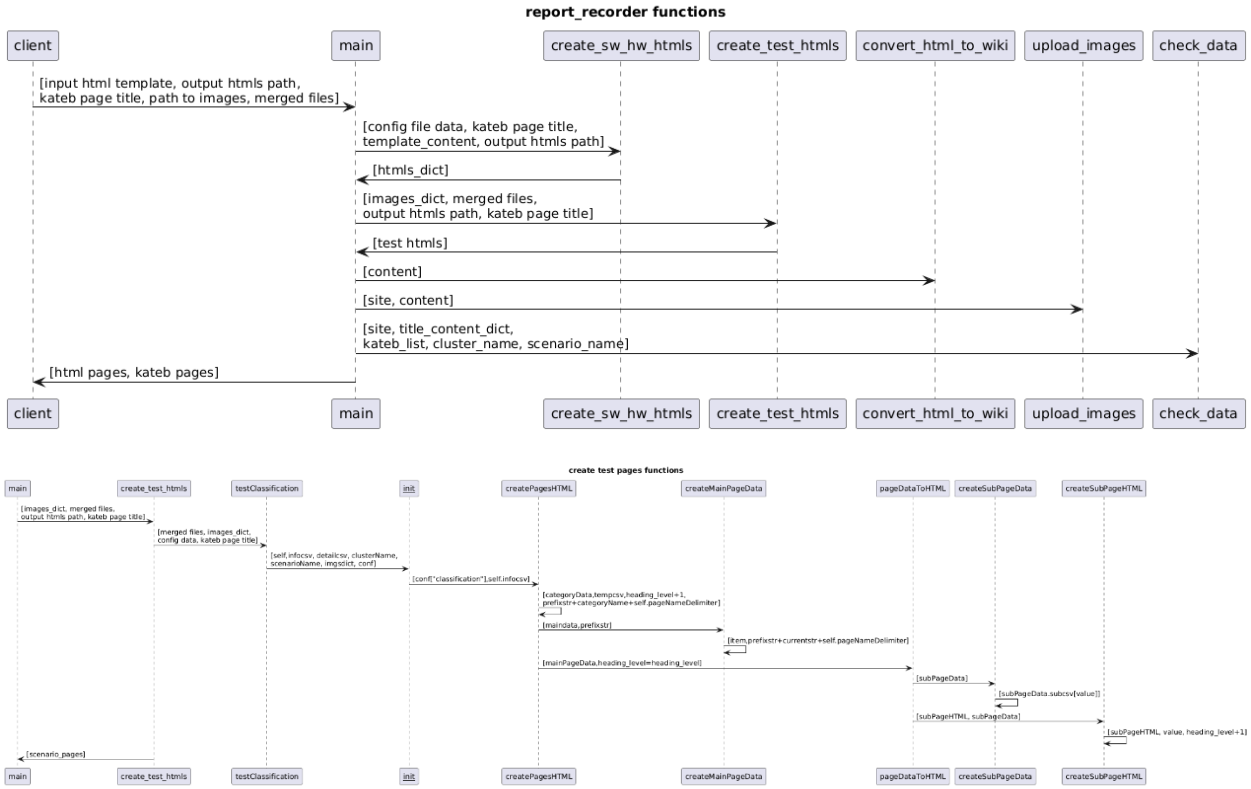
```
def upload_images(site, html_content):
	# Convert dominate document to a string if needed
	image_paths = [img['src'] for img in soup.find_all('img') if 'src' in img.attrs]
	for image_path in image_paths:
		# Upload each image to the wiki
		success = file_page.upload(image_path, comment=f"Uploaded image '{image_filename}' by KARA")
```

۸ـ class testClassification: در این کلاس و توابع و کلاس های زیر مجموعه آن صفحه اصلی تست و زیر صفحه های آن همراه با تصاویر مخصوص هر یک ساخته می‌شوند.

```
class testClassification:
	class subdf
	class subpage
	class mainPage
	def __init__(self,infocsv,detailcsv,clusterName,scenarioName,imgsdict, conf) -> None:
		# read config file data
		# process csv files
		# Convert each row to a tuple and count occurrences
		# make main page data
	def createcsvinfo(self, subdfinstance):
		# read and process csv file
	def divider(self, maindata):
		# create threshold for test per page
	def createMainPageData(self, maindata,prefixstr=""):
```

```
                # make data inside main page

        def createSubPageData(self, subPageData: subPage):

                # make data inside subpages

        def createSubPageHTML(self, subPageHTML: dominate.document,
subPageData :subPage, heading_level=2):

                # create subpages html template

        def pageDataToHTML(self,mainPageData, heading_level):

                #return list of dominate.document

                # convert data inside page to html

                return pagesHTML

        def createPagesHTML(self, classification, subinfocsv,
heading_level=3, prefixstr=""):

                # make html of subpages
```

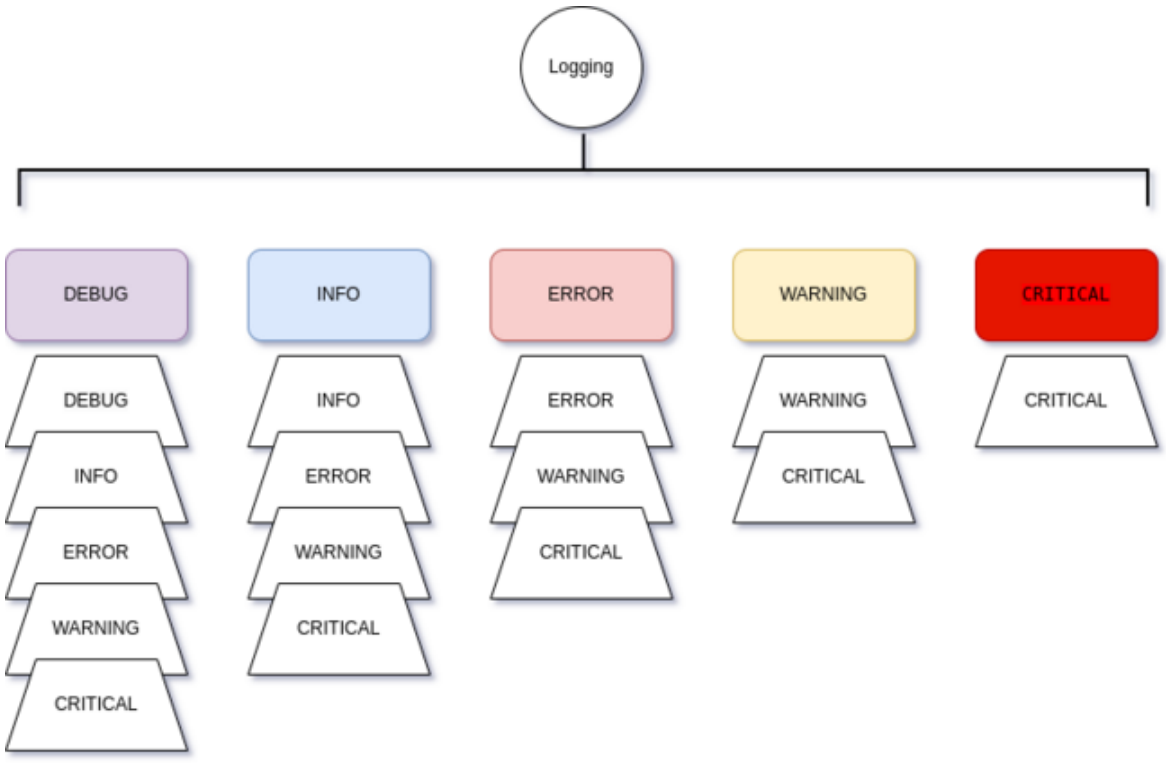
در آخر لیست pagesHTML که شامل همه صفحات است برای ساخت سند استفاده می‌شود.



شکل ۲۷: توابع report_recorder

۲-۴- لاگ ابزارها

مسیر و فایل لاگ تمام ابزار ها یکسان بوده و برابر است با /var/log/kara/all.log و در ابزار های پیشرفته تر و دارای فایل کانفیگ در انتهای فایل این امکان وجود دارد که سطح نمایش لاگ ها را بین مقادیر debug - info - warning - error - critical تغییر داد. برای ابزار های ساده تر و بدون فایل کانفیگ سطح نمایش لاگ debug است . در تصویر ذیل سطح های نمایش لاگ شرح داده شده اند.



شکل ۲۸: لاگ کارا

۲-۵- ابزار تکمیلی (configure)

۲-۵-۱- توسعه ابزار:

این ابزار وظیفه آماده سازی های قبل از نصب کارا را دارد که شامل لینک سازی کد اصلی ابزار ها به محل کتابخانه های پایتون برای import کردن آنها در دیگر ابزار ها و کپی کردن فایل های کانفیگ نمونه به دایرکتوری /etc/kara/ و خارج کردن فایل های بات از حالت فشرده و کپی کردن فایل های پیش نیاز manager است.

```
#!/bin/bash
#### make soft link for tools
# Find the Python directory under /usr/local/lib/
# Find the "kara" directory starting from the current directory
# Array of script names
# Loop through each script and create a symbolic link in the
destination directory
### copy config files to /etc/kara
# Source directory where your files are located
# Destination directory where you want to copy the files
# Check if files are successfully moved
### unzip pywikibot
### copy user-config.py to manager dir
### install dependency
```

فصل ۳

نتیجه گیری

۳-۱- نتیجه گیری:

طبق بررسی و آزمایشات انجام شده استفاده از کارا موجب به حداقل رسیدن خطای انسانی، صرفه جویی در زمان، کاهش نیاز به نیروی انسانی و همچنین کاهش هزینه محصول هیولا خواهد شد.

در پروژه‌های اصلی و عملیاتی محصول هیولا، نیاز است که سرورها و کلاستر موجود برای هر پروژه به طور متوسط دو هفته مورد تست و نظارت قرار گیرند. این بررسی‌ها برای افزایش کارایی و مشخص کردن گلوگاهای موجود ضروری هستند.

با استفاده از کارا در ۴ پروژه عملیاتی محصول هیولا در مجموع بیش از ۲۰۰۰ تست کارایی انجام شده که اطلاعات مربوط به ۱۸۱۶ عدد از این تست‌ها به همراه گزارش وضعیت سخت‌افزار و نرم‌افزار کلاستر توسط این ابزار در سامانه مستندات ثبت شده است. با توجه به تجربه‌های عملیاتی قبلی، در مدت زمان مشابه و بدون استفاده از کارا تعداد تست‌ها و گزارش‌ها رقمی کمتر از ۶۰ عدد است.

نیازهای رفع شده توسط کارا:

- ایجاد انواع کانفیگ برای اجرای تست های مختلف
- امکان بازیابی و یا انتقال اطلاعات پشتیبان‌گیری شده
- مستندسازی گزارش های وضعیت هیولا در سامانه مستندات (کاتب)
- تجمیع داده های به دست آمده از وضعیت هیولا و پردازش و تحلیل آنها
- ذخیره نتایج تست و یا رخداد به صورت جداول csv و گراف و مستند سازی
- اجرای تست ها و workload به صورت خودکار و دریافت نتایج تست و دسته بندی آنها
- مدیریت استفاده از ابزارهای مورد نیاز و اجرای سناریوهای مختلف برای اهداف مشخص
- پشتیبان‌گیری از داده های دیتابیس (influxdb) و پیکربندی‌های سامانه در بازه های زمانی مشخص

۳-۲- کارهای آتی:

به طور کلی نیازمندی‌های اولیه نسخه دوم کارا در دو دسته اصلی و فرعی زیر قرار داده شده‌اند:

دسته اصلی:

- اصلاح ساختار و ماژولار کردن کد ابزارها
- بازتعریف ماژول‌ها در قالب playbook های ansible
- قابلیت کلاسترینگ برای اجرای همزمان کارا روی چندین سرور (ارسال همزمان بار از روی چندین کلاینت به کلاستر هیولا)
- امکان اجرای بنچمارک برای دیسک‌ها(FIO)
- بهبود و اصلاح ویژگی‌های آنالایزر
- نوشتن تست‌های لازم (unit test) برای کدهای اصلاح شده
- مشخص کردن پارامترهای تست (همزمانی و ...) بصورت خودکار با توجه به منابع سخت‌افزاری موجود در کلاستر

دسته فرعی:

- استفاده از message passing در بخش‌هایی مانند گرفتن بکاپ در بکگراند
- مشخص کردن میزان تکرار تست‌ها در کانفیگ manager (هر تست چندبار گرفته شود و سپس میانگین متریک‌ها محاسبه شود)
- فیلتر کردن transformation با توجه به رینگ
- تکمیل لاگ ابزارها
- بررسی ساختار فعلی ابزارها و اصلاح آن در صورت لزوم (مانند ssh زدن به سرور local و دسترسی‌های امنیتی ابزار)
- مشخص کردن پارامترهای کانفیگ OS (مد نظر کاربر) که در صورت متفاوت بودن در تست‌های یک سناریو، در فایل merge-info آورده شوند.
- اضافه کردن قابلیت دسته‌بندی اتوماتیک (اضافه کردن Auto و امکان فیلتر کردن با regex به ابزارreport-recorder)
- اضافه کردن قابلیت خلاصه‌نویسی دسته‌بندی درreport-recorder
- اضافه کردن خلاصه infoCSV به ابتدای سند اصلیreport-recorder

مراجع

1. https://opengit.ir/smartlab/kara
2. https://kateb.burna.ir/r/6B
3. https://github.com/intel-cloud/cosbench
4. https://wiki.openstack.org/wiki/Swift
5. https://grafana.com/docs/grafana/latest/
6. https://github.com/influxdata/influxdb