دانشگاه صنعتي اصفهان

دانشکده برق و کامپیوتر

گزارش کارآموزی

نام و نام خانوادگي کارآموز: مسیح تنورساز

شماره دانشجویی: 40006133

استاد کارآموزی: دکتر امیر اخوان بی تقصیر

سرپرست کارآموزی: جناب آقای حمیدرضا نصر

محل کارآموزی: شرکت پیام پرداز

آدرس: اصفهان، خیابان پروین، نبش خیابان عسکریه روبه روی بانک سپه

تلفن: 09133110900

تاریخ شروع : 16/4/1403 **تاریخ پایان:** 23/6/1403

به نام خدا

فرم شماره ۱
گزارش خلاصه‌ای از فعالیتهای هفتگی

نام و نام خانوادگی کارآموز: مسیح تمزرسار شماره دانشجویی: ٤٠٠٠٦١٣٤

گرایش تحصیلی: IT دانشکده: کامپیوتر (۱۷)

هفته اول از تاریخ: ۴/۱۶ لغایت تاریخ: ۴/۲۲	تکمیل اولیه کارآموزی : یادگیری زبان Golang - پیاده سازی یک برنامه web ساده و استخراج telemetry ها از آن
هفته دوم از تاریخ: ۴/۳۰ لغایت تاریخ: ۵/۵	① : ادامه روند یادگیری زبان Golang - آشنایی با opentelemetry برای استخراج log، trace و metric monitoring یک web app ساده - پیاده سازی یک app برای انداختن تاس و استخراج telemetry های آن
هفته سوم از تاریخ: ۵/۶ لغایت تاریخ: ۵/۱۲	① اتمام یادگیری زبان Golang ② آشنایی با Vault server، Vault - پیاده سازی یک application با ۲ حالت استفاده از Vault و ذخیره سازی گواهی ها درون حافظه local
هفته چهارم از تاریخ: ۵/۱۳ لغایت تاریخ: ۵/۱۹	② ادامه پیاده سازی Vault application با استفاده از زبان Go - آغاز یادگیری docker

نام و نام خانوادگی سرپرست کارآموزی: حمیدرضا ناصر

امضاء سرپرست

به نام خدا

فرم شماره ۱ (ادامه)
گزارش خلاصه‌ای از فعالیتهای هفتگی

نام و نام خانوادگی کارآموز: مسیح تنورساز شماره دانشجویی: ۴۰۰۰۹۱۴۴۴

گرایش تحصیلی: IT دانشکده: کامپیوتر (۱۷)

هفته پنجم از تاریخ: ۵/۲۰ لغایت تاریخ: ۵/۲۶	۱) ادامه روند یادگیری و استفاده از docker - dockerize کردن Vault app و کار با image و container آن
هفته ششم از تاریخ: ۵/۲۷ لغایت تاریخ: ۶/۲	۲) آشنایی با Service mesh - proxy server استفاده از MoSN به عنوان proxyserver - خواندن documentهای MoSN
هفته هفتم از تاریخ: ۶/۳ لغایت تاریخ: ۶/۹	۳) کار با MoSN و تست کردن آن برای • http proxy • https proxy • tcp proxy پیاده سازی MoSN و ساخت image آن
هفته هشتم از تاریخ: ۶/۱۷ لغایت تاریخ: ۶/۲۴	۴) آشنایی با معماری MoSN - Config های آن بعنت cloud native بودن آن به عنوان یک پروژه open source - document کردن پروژه

نام و نام خانوادگی سرپرست کارآموزی: حمیدرضا ناصر

امضاء سرپرست

بیام پرداز
شرکت مهندسی رایانه

به نام خدا

فرم شماره ۲
گزارش سرپرست کارآموزی

نام و نام خانوادگی سرپرست کارآموزی: حمیدرضا صفر سمت: مدیر محصول سدرا
نام واحد صنعتی: شرکت پیام پرداز

نام و نام خانوادگی کارآموز: مسیح تنورساز شماره دانشجویی: ۴۰۰۰۶۱۴۳
گرایش تحصیلی: IT دانشکده: کامپیوتر (۱۷)

ردیف	اظهار نظر سرپرست کارآموز	ضعیف	متوسط	خوب	عالی
۱	رعایت نظم و ترتیب و انضباط در محیط کار				✓
۲	میزان علاقه و همکاری با دیگران				✓
۳	علاقه به فراگیری مطالب علمی و فنی				✓
۴	پیگیری وظایف و میزان پشتکار			✓	
۵	ارزش پیشنهادات کارآموز در جهت بهبود کار				✓
۶	کیفیت گزارشهای کارآموزی (حداقل فرمهای شماره ۱)				✓
۷	میزان بهره گیری از امکانات موجود جهت ارتقاء توانایی علمی و فنی				✓

تعداد روزهای مرخصی: ــــــ تعداد روزهای غیبت: ــــــ

پیشنهادات سرپرست کارآموزی جهت بهبود دورهٔ کارآموزی:

ــــــــــــ

تاریخ ۱۴۰۳/۶/۲۸

امضاء سرپرست کارآموزی

پیام پرداز
شرکت مهندسی ارتباطی

فرم شماره 3

نظرات و پیشنهادات

(در پایان دوره تکمیل شود)

نام و نام خانوادگي كارآموز: مسیح تنورساز شماره دانشجويي: 40006133

رشته تحصيلي: كامپيوتر دانشکده برق و کامپیوتر

شرح نظرات و پيشنهادات:

1- در مورد دوره کارآموزی و مراحل مختلف آن: درباره نحوه شروع آن ابهامت بسیاری وجود دارد که حتی با شرکت کردن در جلسات تدارک دیده شده هنوز هم برطرف نمیشوند. تاریخ پایان آن و نحوه ارائه گزارش نهایی اطلاع رسانی نمیشود. در حالت کلی کمی کاستی برای اطلاع رسانی درباره این دوره از جانب دانشگاه وجود دارد.
2- در مورد امور پژوهشي، فني و توليدي محل كارآموزي: بسته به اینکه محل کارآموز کجا باشد میتواند متفاوت باشد. در شرکت پیام پرداز به دلیل آنکه تعداد زیادی کارآموز در این تابستان گرفته بودند کمی کاستی ها به چشم میخورد. به عنوان مثال بعضا کارآموزان به رشته و کاری متفاوت با آنچه که اعلام آمادگی کرده بودند برایش ورود کردند. از نظر فنی بسته به سرپرست کارآموزی ممکن است پروژه‌های بعضا کم اهمیت به دانشجو داده شود.

توجه: علاوه بر ارائه فرم نظرات و پيشنهادات در پيوست گزارش تفصيلي، در صورت تمايل يك كپي از اين فرم را به دفتر ارتباط با صنعت دانشكده تحويل نماييد.

امضاء كارآموز

فهرست مطالب

چکیده

این گزارش به بررسی و تحلیل پروژه‌های مختلف نرم‌افزاری و تکنولوژیکی که شامل توسعه اپلیکیشن‌های وب با استفاده از Golang، استخراج اطلاعات مفید عملکردی اپلیکیشن با استفاده از OpenTelemetry، پیاده‌سازی سیستم‌های مدیریت اسرار با HashiCorp Vault، مباحث داکر و داکر ایمیج‌ها و بررسی عملکرد MOSN به عنوان یک پروکسی می‌باشد، می‌پردازد. همچنین، درباره پروکسی‌ها و معماری سرویس مش نیز توضیحات مختصری ارائه خواهد شد. تمامی این موارد موضوعاتی اند که طی سه ماه دوران کارآموزی مورد کاوش، پیاده سازی و مطالعه قرار گرفته اند.

فصل اول

معرفی محل کارآموزی

1-1- مقدمه

شرکت مهندسی ارتباطی پیام پرداز در سال ۱۳۷۵ با هدف ارائه خدمات تخصصی در زمینه امنیت اطلاعات و ارتباطات تأسیس گردید. هسته اصلی تشکیل‌دهنده شرکت، ترکیبی از زبده‌ترین پژوهشگران و کارشناسان رمزنگاری و امنیت اطلاعات بود که تا قبل از سال ۱۳۷۵ در قالب حوزه مخابرات امن جهاد دانشگاهی صنعتی اصفهان به فعالیت‌های تحقیقاتی اشتغال داشتند.

شرکت طی سنوات گذشته توانسته است با جذب برجسته‌ترین فارغ‌التحصیلان و نخبگان دانشگاهی و با افزایش توانمندی‌ها و قابلیت‌های تخصصی خود، نقش مؤثری در اجرای پروژه‌های تحقیقاتی و ساخت محصولات انحصاری بازار امنیت فضای تبادل اطلاعات (افتا) کشور ایفا کند.

درحال حاضر دامنه فعالیت شرکت پیام‌پرداز، کلیه حوزه‌های مشاوره، طراحی و اجرا را در برگرفته و از طرح‌های پژوهشی همچون طراحی و تحلیل الگوریتم‌های رمزنگاری تا پروژه‌های پیاده‌سازی نرم‌افزاری و سخت‌افزاری گسترش یافته است.

2-1- معرفی برخی محصولات شرکت

شرکت پیام پرداز، که در اصفهان واقع شده است، محصولات متعددی در حوزه امنیت اطلاعات ارائه می‌دهد. این محصولات عمدتاً برای حفاظت از شبکه‌ها، مدیریت دسترسی‌ها، و رمزنگاری اطلاعات طراحی شده‌اند. لیست زیر شامل برخی از محصولات کلیدی این شرکت به همراه راهکارهایشان است:

1. **توکن امنیتی کیا**
   - **راهکار**: مدیریت امن دسترسی‌ها و احراز هویت برای کاربران.
2. **رمزکننده دیسک ارگ**
   - **راهکار**: رمزنگاری دیسک‌های سخت برای جلوگیری از دسترسی غیرمجاز به اطلاعات ذخیره‌شده.
3. **رمزکننده فایل پاس**
   - **راهکار**: رمزنگاری فایل‌ها برای حفاظت از داده‌ها در برابر دسترسی غیرمجاز.
4. **ورود امن به ویندوز (کیان)**
   - **راهکار**: ایمن‌سازی فرآیند ورود به سیستم‌عامل ویندوز با استفاده از احراز هویت چند عاملی.
5. **دروازه امنیت شبکه (نارین)**
   - **راهکار**: فراهم‌کردن دروازه امنیتی برای کنترل و حفاظت از ترافیک شبکه.
6. **احراز اصالت چندعاملی (سامان MFA)**
   - **راهکار**: استفاده از چند عامل برای احراز هویت امن کاربران در سامانه‌های حساس.
7. **پایش امنیت شبکه (Ravin SIEM)**
   - **راهکار**: مدیریت رخدادهای امنیتی و تحلیل ترافیک برای شناسایی و واکنش به تهدیدات سایبری.
8. **تشخیص نفوذ نسل جدید (Ravin NGIDS)**
   - **راهکار**: تشخیص نفوذهای سایبری پیشرفته و جلوگیری از حملات.
9. **تحلیل رفتار سطح میزبان (Ravin EDR)**
   - **راهکار**: تحلیل رفتارهای مشکوک در سطح میزبان برای شناسایی تهدیدات داخلی و خارجی.

این محصولات بخشی از راهکارهای امنیتی جامع شرکت پیام پرداز هستند که به منظور ایمن‌سازی اطلاعات و شبکه‌ها در سازمان‌ها و شرکت‌های بزرگ ارائه شده‌اند. تخمین کارکنان این شرکت که فعال در شعب مختلف آن در سرتاسر کشور می‌باشند چیزی در حدود 130 نفر می‌باشد.

3-1- معرفی محصول سدرا

محصول سدرا از شرکت پیام پرداز یک راهکار مبتنی بر ابر است که در حوزه زیرساخت ابری و امنیت فعالیت می‌کند. این محصول به کسب‌وکارها امکان می‌دهد تا با استفاده از زیرساخت‌های ابری، داده‌ها و فعالیت‌های کارکنان خود را با امنیت بالا مدیریت، تنظیم و تحلیل کنند. سدرا به گونه‌ای طراحی شده که فرآیندهای کاری در محیط‌های دورکاری و ابری به صورت امن و قابل مدیریت انجام شوند و از مکانیزم‌های نوین امنیتی مانند Zero-Trust برای کنترل دسترسی استفاده می‌کند.

فصل دوم

کارهای انجام شده در دوره کارآموزی

2-1- مقدمه

در دوره کارآموزی، تمرکز من بر تسک‌های متنوع و مختلفی بودند که توسط جناب آقای حمیدرضا نصر مدیرمحصول سدرا به من محول میشدند. در قسمت‌های بعدی به توضیح هریک از این تسک‌ها پرداخته خواهد شد.

2-2- پیاده سازی یک وب اپ ساده و استخراج Telemetry

به عنوان تسک اول،تمرکز من بر پیاده‌سازی یک وب اپلیکیشن ساده با زبان Go و استخراج *telemetry* از آن بود. این پروژه به من این فرصت را داد تا با زبان Go از ابتدا تا پیاده‌سازی نهایی آشنا شوم و در کنار آن، مفاهیم پیشرفته‌تری مثل جمع‌آوری داده‌های عملکردی (Telemetry) با استفاده از OpenTelemetry را به کار بگیرم.

آشنایی با Go

در ابتدا به یادگیری اصول پایه‌ای زبان Go پرداختم. این زبان به خاطر سادگی و عملکرد بالا، انتخاب مناسبی برای توسعه برنامه‌های وب است. در این مرحله، من با مفاهیم پایه‌ای مثل توابع، متغیرها، ساختارهای شرطی، نحوه مدیریت کانال‌ها (Channels) و برنامه نویسی همروند آشنا شدم.

پیاده‌سازی ساختار اصلی اپلیکیشن

برای شروع پروژه، یک وب سرور ساده با استفاده از کتابخانه `net/http` در Go نوشتم. این وب سرور مسیرهایی (routes) را ایجاد می‌کرد که به درخواست‌های HTTP پاسخ می‌دادند. به عنوان مثال، در مسیر `/roll-dice`، کاربر با ارسال درخواست یک عدد تصادفی شبیه به انداختن تاس دریافت می‌کرد. این عدد تصادفی را با استفاده از `rand.Intn` تولید کردم.

پیاده‌سازی اپلیکیشن Roll Dice

پس از تنظیم اولیه سرور، من اپلیکیشنی طراحی کردم که در هر درخواست کاربر یک عدد تصادفی (مثل انداختن تاس) تولید می‌کرد. برای این کار، ابتدا از کتابخانه `rand` برای تولید اعداد تصادفی استفاده کردم و سپس خروجی آن را به کاربر برگرداندم. این بخش به من کمک کرد تا با نحوه مدیریت درخواست‌ها و پاسخ‌ها در Go بیشتر آشنا شوم.

استفاده از OpenTelemetry برای جمع‌آوری Telemetry

پس از پیاده‌سازی اپلیکیشن، به مبحث مهمتر و هدف اصلی این تسک یعنی جمع‌آوری داده‌های عملکردی (Telemetry) پرداختم. OpenTelemetry یک ابزار عالی برای این منظور است که به توسعه‌دهندگان اجازه می‌دهد داده‌های مربوط به عملکرد، خطاها و درخواست‌های ورودی را جمع‌آوری کنند. با نصب کتابخانه OpenTelemetry و تنظیم آن در پروژه، توانستم داده‌های مرتبط با زمان پاسخگویی هر درخواست و دیگر اطلاعات مربوط به عملکرد اپلیکیشن را استخراج کنم.

جمع‌آوری و نمایش داده‌های Telemetry

داده‌های جمع‌آوری‌شده از طریق OpenTelemetry به ابزارهای تحلیل مانند Jaeger ارسال می‌شدند. این ابزارها به من اجازه دادند که تحلیل دقیقی از ترافیک و عملکرد اپلیکیشن داشته باشم. این بخش به من کمک کرد تا بفهمم که چگونه می‌توان از داده‌های واقعی برای بهینه‌سازی و بهبود عملکرد برنامه استفاده کرد.

جمع‌بندی و نتیجه‌گیری

در پایان، این پروژه به من کمک کرد تا درک عمیقی از زبان Go و ابزارهای مرتبط با جمع‌آوری داده‌های عملکردی پیدا کنم. توانستم با استفاده از OpenTelemetry داده‌های مفیدی از عملکرد اپلیکیشن جمع‌آوری کنم که به بهبود کیفیت و عملکرد برنامه‌های کلان با تعداد بسیار زیاد مشتری‌ها کمک می‌کند.

2-3- پیاده سازی vault و ذخیره اطلاعات در دیسک محلی

در دومین تسک دوره کارآموزی، تمرکز من روی پیاده‌سازی یک اپلیکیشن با استفاده از Vault یا ذخیره‌سازی اطلاعات در دیسک محلی بود. این پروژه به من امکان داد تا با مفاهیم مرتبط با امنیت، مدیریت رمزها و روش‌های مختلف ذخیره‌سازی آشنا شوم. پروژه‌ای که پیاده‌سازی کردم به نوعی مشابه پروژه *Trasa* بود و نحوه استفاده از HashiCorp Vault را در آن بررسی کردم.

آشنایی با HashiCorp Vault

HashiCorp Vault یکی از ابزارهای قدرتمند برای مدیریت رمزها و دسترسی‌های امن به داده‌های حساس است. ابتدا با مفاهیم پایه‌ای Vault آشنا شدم، از جمله اینکه چگونه این ابزار می‌تواند به صورت امن رمزها را ذخیره کند، احراز هویت انجام دهد و دسترسی‌ها را مدیریت نماید.[1]

پیاده‌سازی اپلیکیشن با Vault

در این مرحله، هدف من پیاده‌سازی اپلیکیشنی بود که از Vault برای ذخیره‌سازی و مدیریت اطلاعات استفاده کند. ابتدا یک سرور Vault راه‌اندازی کردم و با استفاده از APIهای Vault، اپلیکیشن به گونه‌ای طراحی شد که رمزها و اطلاعات حساس از طریق Vault مدیریت شوند.

در این پروژه از دو روش برای مدیریت رمزها استفاده شد:

- **ذخیره‌سازی در Vault**: اطلاعات حساس مانند رمزهای عبور در Vault ذخیره شدند و با استفاده از توکن‌ها و مکانیزم‌های امنیتی Vault به این اطلاعات دسترسی پیدا می‌کردیم.
- **ذخیره‌سازی محلی**: در مواردی که نیاز به ذخیره اطلاعات به صورت محلی بود، این داده‌ها در فایل‌های محلی آن هم به صورت رمز شده روی دیسک ذخیره می‌شدند. برای این کار از سیستم فایل ساده Go استفاده شد.

مدیریت دسترسی‌ها و امنیت

در این بخش، به نحوه استفاده از قابلیت‌های امنیتی Vault پرداختم. هر درخواست به Vault باید دارای توکن دسترسی می‌بود و بر اساس سیاست‌های تعریف شده، دسترسی‌های مختلف به اطلاعات مدیریت می‌شد. این سیاست‌ها به صورت دقیق تنظیم شدند تا فقط کاربران و سیستم‌های مجاز به اطلاعات دسترسی داشته باشند.

جمع‌بندی و نتیجه‌گیری

این پروژه به من کمک کرد تا با روش‌های مختلف مدیریت رمزها و اطلاعات حساس آشنا شوم و نحوه پیاده‌سازی یک سیستم امن برای ذخیره‌سازی داده‌ها را به‌خوبی بیاموزم. استفاده از Vault در این پروژه باعث شد تا امنیت داده‌ها به صورت کامل تضمین شود. در محصول سدرا که جزو محصولات ابری و امنیتی شرکت پیام پرداز است چنین موردی پیاده سازی شده است و برای ذخیر اطلاعات حساس مانند رمزها، نشست‌ها و گواهی‌ها مورد استفاده قرار میگیرد.

2-4- داکرایز کردن پروژه vault

برای داکرایز کردن پروژه‌ی مدیریت رمزها با استفاده از Vault و Go، ابتدا باید با Docker و نحوه کار آن آشنا می‌شدم. برای این کار، از یک دوره‌ی آموزشی به نام The Ultimate Docker Course استفاده کردم که در آن مباحث مقدماتی و پیشرفته Docker به صورت کامل پوشش داده شده است. در طول این دوره، یاد گرفتم که چگونه کانتینرها را ایجاد و مدیریت کنم، از Docker Compose برای مدیریت سرویس‌های چندگانه استفاده کنم، و کانتینرها را شبکه‌بندی کنم.[2]

یادگیری Docker

در مرحله‌ی یادگیری، مباحث پایه‌ای Docker مانند:

- **ساخت کانتینرها**: از تصاویر آماده یا تعریف تصاویر دلخواه با استفاده از Dockerfile.
- **مدیریت شبکه‌ها و ولوم‌ها**: برای ارتباط و اشتراک‌گذاری داده‌ها بین کانتینرها.
- **Docker Compose**: برای مدیریت چندین سرویس همزمان در پروژه‌ها.

این مباحث پایه به من کمک کرد تا با ساختار داکر آشنا شوم و از آن برای پروژه‌ی خود استفاده کنم.

Dockerizing پروژه‌ی Vault

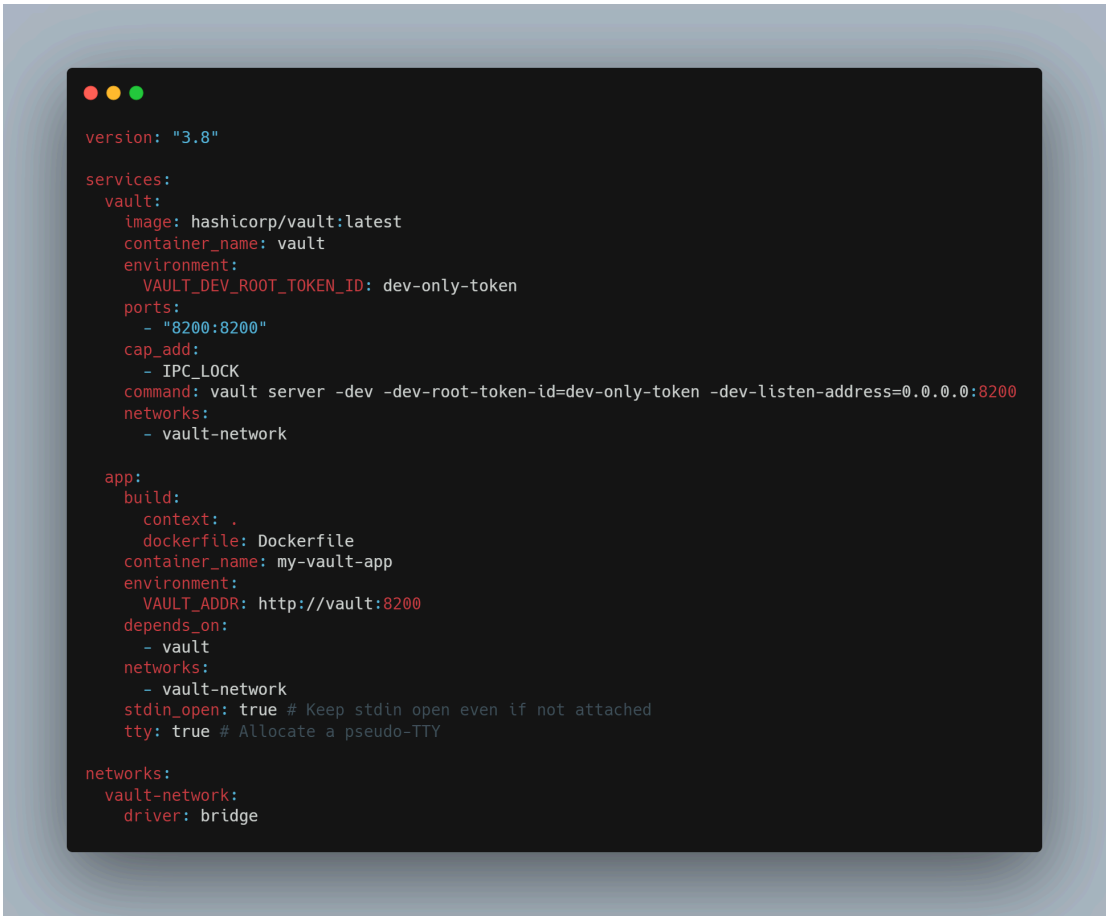
پس از یادگیری مباحث پایه، پروژه‌ای را که شامل یک سرویس Vault و یک اپلیکیشن Go بود، داکرایز کردم. هدف از داکرایز کردن این پروژه، اجرای آن به صورت مجزا در کانتینرها و مدیریت آسان‌تر وابستگی‌ها بود.

تنظیم فایل Docker Compose

فایل `docker-compose.yml` پروژه برای مدیریت دو سرویس اصلی تنظیم شده است:

1. **سرویس Vault**: این سرویس از تصویر آماده‌ی `hashicorp/vault` استفاده می‌کند و با استفاده از یک **توکن توسعه (dev token)** به صورت محلی راه‌اندازی می‌شود. برای ایمن‌سازی این سرویس، از تنظیمات `cap_add: IPC_LOCK` برای جلوگیری از قرارگیری اطلاعات حساس در حافظه استفاده شد. همچنین پورت 8200 برای دسترسی به Vault از خارج از کانتینر تنظیم شده است.
2. **سرویس اپلیکیشن**: اپلیکیشنی که در Go نوشته شده و از Vault برای ذخیره و بازیابی اطلاعات استفاده می‌کند. این اپلیکیشن به کمک Dockerfile ساخته شده و تنظیمات لازم برای اتصال به سرویس Vault از طریق متغیر محیطی `VAULT_ADDR` انجام شده است.

فایل `docker-compose.yml`:



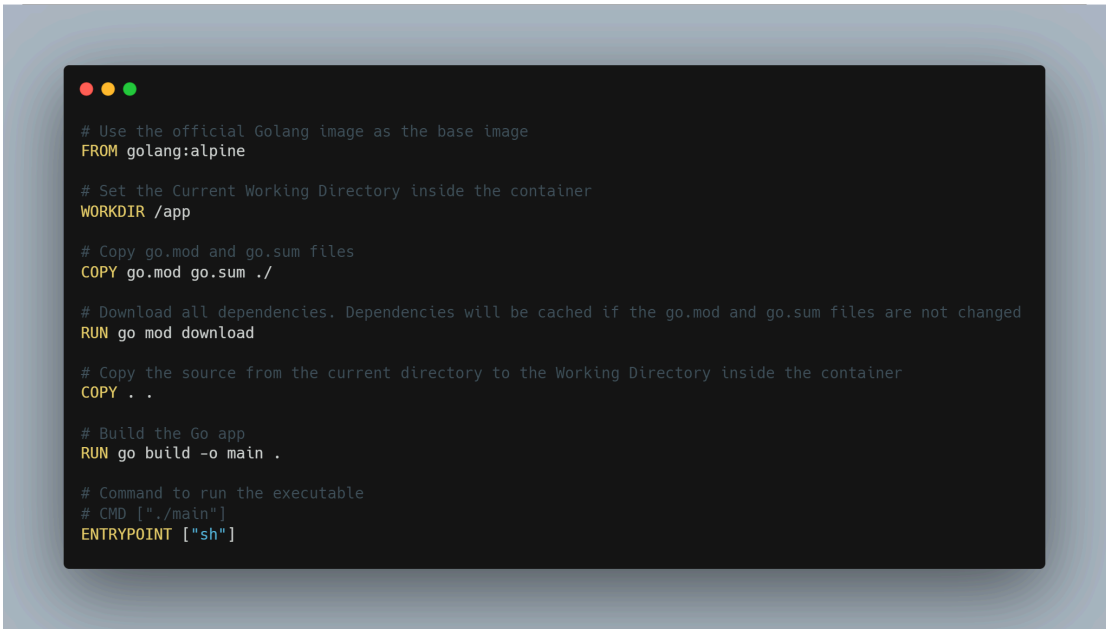
```yaml
version: "3.8"

services:
  vault:
    image: hashicorp/vault:latest
    container_name: vault
    environment:
      VAULT_DEV_ROOT_TOKEN_ID: dev-only-token
    ports:
      - "8200:8200"
    cap_add:
      - IPC_LOCK
    command: vault server -dev -dev-root-token-id=dev-only-token -dev-listen-address=0.0.0.0:8200
    networks:
      - vault-network

  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: my-vault-app
    environment:
      VAULT_ADDR: http://vault:8200
    depends_on:
      - vault
    networks:
      - vault-network
    stdin_open: true # Keep stdin open even if not attached
    tty: true # Allocate a pseudo-TTY

networks:
  vault-network:
    driver: bridge
```

Dockerfile اپلیکیشن Go

برای ساخت تصویر اپلیکیشن Go، فایل `Dockerfile` را به صورت زیر تنظیم کردم:

1. استفاده از تصویر پایه‌ی golang:alpine برای ساخت و اجرای اپلیکیشن.

2. کپی کردن کدهای پروژه به داخل کانتینر.
3. نصب وابستگی‌ها و کامپایل کدها.
4. اجرای اپلیکیشن پس از آماده‌سازی.



```
# Use the official Golang image as the base image
FROM golang:alpine

# Set the Current Working Directory inside the container
WORKDIR /app

# Copy go.mod and go.sum files
COPY go.mod go.sum ./

# Download all dependencies. Dependencies will be cached if the go.mod and go.sum files are not changed
RUN go mod download

# Copy the source from the current directory to the Working Directory inside the container
COPY . .

# Build the Go app
RUN go build -o main .

# Command to run the executable
# CMD ["./main"]
ENTRYPOINT ["sh"]
```

جمع‌بندی و نتیجه‌گیری

با استفاده از Docker، توانستم پروژه را در دو سرویس مجزا (Vault و اپلیکیشن Go) اجرا کنم. این کار به من کمک کرد تا به صورت راحت‌تر و سریع‌تر برنامه‌ها را تست کنم و مطمئن شوم که ارتباطات بین سرویس‌ها به‌درستی کار می‌کنند. علاوه بر این، استفاده از Docker و Docker Compose مدیریت وابستگی‌ها و محیط اجرا را بسیار ساده‌تر کرد.[3]

5-2- نحوه کانفیگ و کار با MOSN

در این تسک تمرکز من بر کانفیگ و کار با MOSN [1] بود. MOSN یک پروکسی cloud-native است که برای کار با پروتکل‌های مختلف شبکه‌ای طراحی شده است. در این فرآیند، علاوه بر تست و راه‌اندازی MOSN برای هر کدام از پروتکل‌های TCP، HTTPS، HTTP و UDP، با معماری آن و دلیل اینکه یک ابزار cloud-native است نیز آشنا شدم. همچنین برای اجرای پروژه، MOSN را داکرایز کردم و از مستندات موجود در سایت mosn.io برای این کار استفاده کردم.

[1] Modular Open Smart Network

آشنایی با MOSN

MOSN یک پروکسی cloud-native است که برای مدیریت ترافیک در سرویس‌های مختلف استفاده می‌شود. این پروکسی به خصوص در محیط‌های microservice و معماری‌های مبتنی بر cloud بسیار کارآمد است. یکی از ویژگی‌های مهم MOSN، پشتیبانی از پروتکل‌های متنوع شبکه است و می‌تواند در نقش‌های مختلف مانند edge proxy، ingress proxy یا service mesh عمل کند.

چرا MOSN cloud-native است؟

1. **مقیاس‌پذیری**: MOSN به عنوان یک پروکسی cloud-native طراحی شده است، یعنی قابلیت استفاده در محیط‌های توزیع‌شده را دارد و به راحتی می‌تواند مقیاس‌پذیری بالایی داشته باشد.
2. **ماژولار بودن**: معماری ماژولار MOSN به آن امکان می‌دهد تا به‌راحتی با نیازهای مختلف شبکه‌ای سازگار شود.
3. **پشتیبانی از انواع پروتکل‌ها**: MOSN از پروتکل‌های TCP، HTTPS، HTTP، و حتی پروتکل‌های پیچیده‌تر مانند gRPC پشتیبانی می‌کند.

داکرایز کردن MOSN

برای راه‌اندازی MOSN، ابتدا یک کانتینر Docker برای آن ایجاد کردم. تصویر پایه‌ای که استفاده کردم golang:alpine بود که سبکی و سرعت بالای اجرای آن برای این کار مناسب است.
مراحل ساخت ایمیج داکر به صورت زیر بودند:

1. **انتخاب تصویر پایه**:
   - با تصویر پایه‌ی `golang:alpine` شروع می‌کنیم.
2. **نصب وابستگی‌ها**:
   - به‌روزرسانی مخازن و نصب وابستگی‌های مورد نیاز مثل `curl`, `bash`, و `gcc`.
3. **تنظیم دایرکتوری کار**:
   - دایرکتوری کار به `go/src/mosn/` تنظیم می‌شود.
4. **کپی فایل‌های محلی به کانتینر**:
   - فایل‌های محلی MOSN به کانتینر کپی می‌شوند.
5. **ساخت MOSN**:
   - با استفاده از MOSN، `make build-local` درون کانتینر ساخته می‌شود.
6. **نصب وابستگی‌های اضافی و تنظیم Zsh**:
   - نصب ابزارهای اضافی مثل `git`, `zsh`, و `openssl` و نصب Oh My Zsh.
7. **ساخت سرور HTTP**:
   - دایرکتوری کار به `http-sample` تغییر می‌یابد و سرور ساخته می‌شود.
8. **اجرای Zsh به‌عنوان شل پیش‌فرض**:
   - کانتینر با Zsh شروع می‌شود.

در نهایت داکرفایل نهایی به صورت زیر آماده می‌شود:

```
# Start with the Golang Alpine base image
FROM golang:alpine

# Install Docker and other dependencies
RUN apk update && \
    apk add --no-cache \
    bash \
    ca-certificates \
    iptables \
    curl \
    make \
    libc6-compat \
    build-base \
    gcc \
    g++


# Set up the working directory for the application
WORKDIR /go/src/mosn

# Copy the local MOSN files into the container
COPY ./mosn /go/src/mosn

# # Build MOSN inside the container
RUN make build-local
WORKDIR /go/src/mosn/cmd/mosn/main
RUN go build .

RUN apk update && \
    apk add --no-cache \
    bash \
    zsh \
    git \
    openssl && \
    # Install Oh My Zsh
    sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)" && \
    # Set Zsh as the default shell
    # Clean up
    apk del git && \
    mv main /go/src/mosn/examples/codes/http-sample
# RUN go run /go/src/mosn/examples/codes/http-sample/server.go
WORKDIR /go/src/mosn/examples/codes/http-sample
RUN go build server.go && apk add nano
# Set environment variables for MOSN
# ENV CONFIG_FILE=/go/src/mosn/build/bundles/v1.6.0/binary/mosn_config.json
# # Expose port 2045 for MOSN or any other port you plan to use
# EXPOSE 2045

# # Start Docker daemon and run a Bash shell for interaction
# CMD ["sh", "-c", "dockerd & bash"]
CMD ["zsh"]
```

راه‌اندازی MOSN به عنوان HTTP Proxy

در این مرحله، MOSN را به عنوان یک HTTP Proxy راه‌اندازی کردم. MOSN قابلیت مسیریابی و مدیریت درخواست‌های HTTP را دارد. تنظیمات مورد نیاز برای HTTP Proxy را در فایل‌های کانفیگ MOSN قرار دادم و پروکسی را راه‌اندازی کردم.

راه‌اندازی MOSN به عنوان HTTPS Proxy

پس از تست موفق HTTP، به سراغ HTTPS Proxy رفتم. در این بخش، نیاز بود تا MOSN با گواهی‌های SSL تنظیم شود تا بتواند درخواست‌های HTTPS را مدیریت کند. تنظیمات مورد نیاز را در کانفیگ MOSN برای فعال‌سازی TLS قرار دادم و این بخش نیز با موفقیت پیاده‌سازی شد.

راه‌اندازی MOSN به عنوان TCP Proxy

در نهایت، MOSN را به عنوان یک TCP Proxy راه‌اندازی کردم. MOSN با پشتیبانی از پروتکل‌های سطح پایین‌تر مانند TCP، به من این امکان را داد تا درخواست‌های TCP را نیز مدیریت کنم. این بخش برای سناریوهایی که پروتکل‌های سفارشی و غیر HTTP استفاده می‌شوند، بسیار مفید بود.

جمع‌بندی و نتیجه‌گیری

در طول این تسک، علاوه بر تست MOSN در حالت‌های مختلف پروکسی HTTPS ،HTTP و TCP، با معماری cloud-native و ماژولار MOSN به خوبی آشنا شدم. با استفاده از Docker، توانستم MOSN را به سرعت داکرایز کرده و به صورت کانتینری تست کنم. این کار به من درک عمیق‌تری از ابزارهای cloud-native و پروکسی‌های مدرن داد.[4]

فصل سوم

گزارش درمورد پروکسی‌ها

3-1- مقدمه

پروکسی‌ها به عنوان میان‌افزارهایی عمل می‌کنند که درخواست‌های کلاینت‌ها را دریافت کرده و آن‌ها را به سرورهای مقصد منتقل می‌کنند. این فناوری‌ها در بهینه‌سازی ترافیک، افزایش امنیت، و مقیاس‌پذیری در معماری‌های میکروسرویس‌ها نقش حیاتی دارند. در این گزارش، به بررسی پروکسی‌ها و به ویژه MOSN خواهیم پرداخت.

3-2- مفهوم پروکسی

پروکسی به معنای "نماینده" است و در زمینه شبکه به عنوان یک واسطه بین کلاینت و سرور عمل می‌کند. پروکسی‌ها می‌توانند درخواست‌ها را فیلتر، مسیریابی، یا بهبود بخشند. مزایای اصلی استفاده از پروکسی‌ها را می‌توان اینچنین نام برد:

- **حفاظت و امنیت**: پروکسی‌ها می‌توانند به عنوان یک لایه حفاظتی عمل کنند و از سرورهای اصلی در برابر حملات محافظت کنند.
- **کاهش تأخیر**: با کش کردن داده‌ها، پروکسی‌ها می‌توانند زمان بارگذاری را کاهش دهند.
- **توزیع بار**: با توزیع ترافیک بین چند سرور، می‌توان به بهبود مقیاس‌پذیری و کارایی کمک کرد.

3-3- انواع پروکسی

انواع مختلفی از پروکسی‌ها وجود دارند که شامل موارد زیر هستند:

- **پروکسی‌های معکوس (Reverse Proxy)**: این نوع پروکسی درخواست‌ها را از کلاینت‌ها دریافت کرده و آن‌ها را به سرورهای مختلف هدایت می‌کند. این پروکسی‌ها معمولاً برای بارگذاری متوازن و امنیت استفاده می‌شوند.
- **پروکسی‌های جلو (Forward Proxy)**: این پروکسی‌ها درخواست‌ها را از کلاینت‌ها به سرورهای مقصد منتقل می‌کنند و به کاربر این امکان را می‌دهند که به محتوای محدود شده دسترسی پیدا کند.

3-4- آشنایی با MOSN

MOSN یک پروکسی مدرن و مقیاس‌پذیر است که بر اساس معماری میکروسرویس‌ها ساخته شده است. این پروکسی به طور خاص برای مدیریت ترافیک شبکه در سیستم‌های توزیع‌شده طراحی شده است.

3-4-1- ویژگی‌های MOSN

- **عملکرد بالا**: MOSN برای بهینه‌سازی عملکرد و کاهش تأخیر طراحی شده است.
- **پشتیبانی از پروتکل‌های مختلف**: این پروکسی از HTTP, HTTPS و TCP پشتیبانی می‌کند.
- **مقیاس‌پذیری**: MOSN به سادگی می‌تواند در مقیاس بزرگ، با هزاران سرور و هزاران کلاینت کار کند.
- **مدیریت ترافیک هوشمند**: MOSN قابلیت‌های پیشرفته‌ای مانند بارگذاری متوازن، مسیریابی هوشمند و نظارت بر ترافیک را ارائه می‌دهد.

3-4-2- استفاده‌ها و کاربردها

MOSN در محیط‌های مختلف مانند:

- **معماری میکروسرویس**: برای مدیریت ترافیک بین میکروسرویس‌ها.
- **کلاود نیتیو**: برای تسهیل ارتباطات در برنامه‌های کلاود نیتیو.
- **پروژه‌های بزرگ**: برای ارائه مقیاس‌پذیری و عملکرد مطلوب.

جمع‌بندی و نتیجه‌گیری

پروکسی‌ها به عنوان یک ابزار مهم در بهینه‌سازی عملکرد شبکه و افزایش امنیت نقش کلیدی دارند. MOSN با ویژگی‌های خاص خود، به عنوان یک پروکسی مدرن، امکان مدیریت مؤثر ترافیک در محیط‌های پیچیده و توزیع‌شده را فراهم می‌آورد. با توجه به نیازهای روزافزون به مقیاس‌پذیری و عملکرد در دنیای دیجیتال، استفاده از چنین ابزارهایی بیش از پیش اهمیت پیدا کرده است.

پیوست

سرویس مش

مقدمه

سرویس مش[2] یک لایه مدیریت ارتباطات بین میکروسرویس‌ها است که به توسعه‌دهندگان امکان می‌دهد تا تعاملات پیچیده‌ی میان سرویس‌ها را به‌سادگی مدیریت کنند. این فناوری به ویژه در معماری‌های میکروسرویس توزیع‌شده کاربرد دارد و می‌تواند به بهبود مقیاس‌پذیری، امنیت و مشاهده‌پذیری کمک کند. در این گزارش، به بررسی مفهوم سرویس مش، اجزا، مزایا و کاربردهای آن خواهیم پرداخت.

۱. مفهوم سرویس مش

سرویس مش به عنوان یک شبکه‌ای از سرویس‌ها عمل می‌کند که به طور خودکار و بدون دخالت توسعه‌دهندگان، ارتباطات و تعاملات بین میکروسرویس‌ها را مدیریت می‌کند. این مدیریت شامل مسیریابی ترافیک، بارگذاری متوازن، امنیت، و نظارت بر ترافیک می‌شود.

۲. اجزای سرویس مش

سرویس مش معمولاً شامل دو بخش اصلی است:

- **کنترل‌گر (Control Plane)**: این قسمت مسئول مدیریت سیاست‌ها و تنظیمات برای سرویس مش است. کنترل‌گر می‌تواند تنظیمات را به تمام نودهای موجود در شبکه ارسال کند.
- **داده‌گر (Data Plane)**: این بخش شامل پروکسی‌هایی است که در کنار هر میکروسرویس قرار دارند و وظیفه‌ی مدیریت ترافیک ورودی و خروجی را بر عهده دارند. این پروکسی‌ها، ارتباطات را بررسی و پردازش می‌کنند.

---

[2] Service Mesh

مراجع

[1] “HashiCorp Vault | Identity-based secrets management.” *HashiCorp*, https://www.hashicorp.com/products/vault. Accessed 22 September 2024.

[2] “The Ultimate Docker Course.” *Code with Mosh*, https://codewithmosh.com/p/the-ultimate-docker-course. Accessed 22 September 2024.

[3] *Docker: Accelerated Container Application Development*, https://www.docker.com/. Accessed 22 September 2024.

[4] “mosn/mosn: The Cloud-Native Network Proxy Platform.” *GitHub*, https://github.com/mosn/mosn. Accessed 22 September 2024.

.