



دانشگاه صنعتی شریف

دانشکده مهندسی برق

پایان نامه کارشناسی ارشد

گرایش سیستم های الکترونیک دیجیتال

طراحی و پیاده سازی یک پلتفرم MLOps به صورت ابری مبتنی بر GPU

نگارنده

ابوالفضل یاریان

استاد راهنما

دکتر متین هاشمی

۱۴۰۳ تیرماه

بسمه تعالی

دانشگاه صنعتی شریف
دانشکده مهندسی برق

پایاننامه کارشناسی ارشد

عنوان: طراحی و پیاده‌سازی یک پلتفرم MLOps به صورت ابری مبتنی بر GPU
نگارش: ابوالفضل یاریان

اعضا هیات داوران:

..... امضاء: دکتر متین هاشمی
..... امضاء: دکتر ...
..... امضاء: دکتر ...
..... امضاء: دکتر ...
..... امضاء: دکتر ...

تاریخ:

تقدیم و قدردانی

این پایان نامه را به پدر و مادر عزیزم تقدیم می کنم که همواره حامی و مشوق من بوده اند. از استاد راهنمای گرامی، دکتر متین هاشمی، بابت راهنمایی ها و حمایت های بی دریغ شان صمیمانه تشکر می کنم. همچنین، قدردانی ویژه ای دارم از دکتر طاها قاسمی به خاطر مشاوره ها و راهنمایی های ارزشمند شان در طول این مسیر و از تمامی دوستان و همکارانی که با همدلی و همراهی مرا در این مسیر یاری دادند نیز بی نهایت سپاسگزارم.

چکیده:

در عصر حاضر، هوش مصنوعی و یادگیری ماشین به یکی از حیاتی‌ترین و پرکاربردترین فناوری‌ها در صنایع مختلف تبدیل شده‌اند. این فناوری‌ها به شرکت‌ها و سازمان‌ها این امکان را می‌دهند تا بهینه‌سازی فرآیندها، پیش‌بینی روندها، و کشف الگوهای پنهان در داده‌ها را با دقت و سرعت بالا انجام دهند. با این حال، بهره‌گیری کامل از قابلیت‌های هوش مصنوعی و یادگیری ماشین نیازمند پیاده‌سازی مؤثر و کارآمد مدل‌های یادگیری ماشین در محیط‌های تولیدی است. MLOps، که ترکیبی از مفاهیم DevOps و ML است، به مدیریت چرخه عمر مدل‌های یادگیری ماشین از توسعه تا استقرار و نگهداری کمک می‌کند. در این تحقیق، به دلیل تحریم‌های بین‌المللی و محدودیت‌های دسترسی به سرویس‌های خارجی مانند Amazon Vertex AI، Google Vertex AI و Microsoft Azure ML و SageMaker، یک پلتفرم بومی MLOps طراحی و پیاده‌سازی شده است. این پلتفرم با استفاده از ابزارهای متن‌باز توسعه داده شده و قابلیت‌های مختلفی از جمله مدیریت چرخه عمر مدل، نظارت بر عملکرد، بهروزرسانی خودکار و تعامل با ابزارهای موجود را فراهم می‌کند. این تحقیق با هدف پاسخ به سوالات تحقیقاتی مرتبط با نیازهای یک پلتفرم MLOps مدرن و امکان‌پذیری پیاده‌سازی آن با استفاده از ابزارهای متن‌باز انجام شده است. نتایج حاصل از پیاده‌سازی و آزمایش‌ها نشان می‌دهند که پلتفرم پیشنهادی نه تنها به نیازهای مدرن پاسخ می‌دهد بلکه استقلال از ارائه‌دهندگان ابر را نیز فراهم می‌آورد. نهایتاً، یافته‌های تحقیق و پیشنهادات برای تحقیقات آینده ارائه شده‌اند.

كلمات کلیدی:

- | | |
|--|---------------------------------|
| Machine Learning Operation (MLOps). | ۱-پیاده‌سازی یادگیری ماشین |
| Continuous Integration (CI). | ۲-ادغام مداوم |
| Continuous Deployment (CD). | ۳-استقرار مداوم |
| Machine Learning Lifecycle Management. | ۴-مدیریت چرخه عمر یادگیری ماشین |
| Open-Source Cloud platform. | ۵-پلتفرم ابری متن‌باز |

فهرست مطالب

۱	۱	۱	مقدمه
۱	۱-۱	۱	تعريف مسئله و بیان چالش‌ها
۳	۲-۱	۳	روش تحقیق
۵	۳-۱	۵	ساختار پایان نامه
۷	۲	۷	مرور مفاهیم پایه
۷	۱-۲	۷	مقدمه
۷	۲-۲	۷	DevOps
۷	۱-۲-۲	۷	تعريف
۸	۲-۲-۲	۸	چرخه کاری
۱۰	۳-۲-۲	۱۰	خط لوله CI/CD
۱۳	۴-۲-۲	۱۳	مزایای متداول‌تری DevOps
۱۴	۳-۲	۱۴	مجازی سازی و کانتینرها
۱۴	۱-۳-۲	۱۴	مجازی سازی
۱۶	۲-۳-۲	۱۶	کانتینرها
۱۹	۳-۳-۲	۱۹	هماننگ سازی کانتینرها (کوبرنتیز)
۲۲	۳	۲۲	مروری بر مفاهیم MLOps
۲۲	۱-۳	۲۲	مقدمه
۲۲	۲-۳	۲۲	تعريف مفاهیم اولیه
۲۳	۱-۲-۳	۲۳	اصول

۲۵	اجزاء	۲-۲-۳
۳۱	نقش ها	۳-۲-۳
۳۲	معماری جامع	۳-۳

۳۹

۴ طراحی پلتفرم MLOps

۳۹	مقدمه	۱-۴
۳۹	سیستم مدیریت پلتفرم	۲-۴
۳۹	مدیریت پیکربندی و فراهم سازی زیرساخت	۱-۲-۴
۴۲	خط لوله CI/CD	۲-۲-۴
۴۳	مخزن کد منبع	۳-۲-۴
۴۴	مخزن مؤلفه ها	۴-۲-۴
۴۵	معماری خوشه کوبرنیز	۳-۴
۴۶	مدیریت داده و مدل	۱-۳-۴
۵۰	شبکه	۲-۳-۴
۵۳	مدیریت کاربران و چندمستاجری	۳-۳-۴
۵۵	نظارت	۴-۳-۴
۵۷	استقرار مدل	۵-۳-۴
۶۱	جريان کاری یادگیری ماشین	۶-۳-۴

۶۴

۵ پیاده سازی و نتایج

۶۴	مقدمه	۱-۵
۶۴	پیاده سازی پلتفرم	۲-۵
۶۴	سیستم مدیریت	۱-۲-۵
۶۷	خوشه کوبرنیز	۲-۲-۵
۷۰	حل مسئله و نتایج	۳-۵
۷۱	مسئله تشخیص ارقام	۱-۳-۵
۷۳	مسئله تحلیل احساسات در بازار سهام	۲-۳-۵

۸۰

۶ نتیجه گیری و پیشنهادات

فهرست جداول

۱۱	نمونه‌هایی از ابزار برای مراحل خاص اتوماسیون خط لوله CI/CD	۱-۲
۶۵	مشخصات سخت افزاری ماشین‌های مدیریت	۱-۵
۶۷	مشخصات سخت افزاری ماشین‌های خوش کوبرنتیز	۲-۵
۷۲	زمان اجرا چرخه یادگیری ماشین در مسئله تشخیص ارقام	۳-۵
۷۳	تست استنتاج مدل مسئله تشخیص ارقام	۴-۵
۷۸	زمان اجرا در مسئله تحلیل احساسات در بازار سهام	۵-۵
۷۸	تست استنتاج مدل مسئله تحلیل احساسات در بازار سهام	۶-۵

فهرست تصاویر

۲	زمان استقرار مدل توسط دانشمندان داده	۱-۱
۹	مراحل DevOps	۱-۲
۱۵	انواع هایپروایزر [۱]	۲-۲
۱۷	تفاوت ماشین مجازی و کانتینر	۳-۲
۱۸	معماری لایه‌ای تصویر داکر	۴-۲
۲۰	مولفه‌های یک خوشه کوبرنتیز	۵-۲
۲۶	خط لوله در Apache Airflow	۱-۳
۲۷	انباره ویژگی	۲-۳
۳۱	نقش‌ها و اشتراکات آن‌ها در پارادایم MLOps	۳-۳
۳۲	معماری جامع [۲] MLOps	۴-۳
۴۱	معماری Ansible	۱-۴
۴۲	استقرار مبتنی بر Pull	۲-۴
۴۳	استقرار مبتنی بر Push	۳-۴
۴۷	نحوه کار PVC و PVC در خوشه کوبرنتیز	۴-۴
۴۸	معماری MinIO	۵-۴
۵۲	معماری Istio	۶-۴
۵۴	رونده احراز هویت	۷-۴
۵۶	معماری بخش نظارت	۸-۴
۶۰	معماری سطح داده در KServe	۹-۴
۶۲	معماری Kubeflow Pipelines	۱۰-۴
۶۶	مخازن مؤلفه در Nexus	۱-۵
۶۶	خط لوله‌های CI/CD در Jenkins	۲-۵
۶۷	مخازن کد در Gitlab	۳-۵
۶۹	وضعیت مؤلفه‌ها در خوشه کوبرنتیز	۴-۵
۷۰	میزان مصرف منابع سخت‌افزاری پلتفرم بدون بار	۵-۵
۷۰	داشبورد پلتفرم MLOps	۶-۵

۷۱	نمونه داده مجموعه دادگان MNIST	۷-۵
۷۳	گراف استنتاج مدل در مسئله تشخیص ارقام (Grafana (داشبورد	۸-۵
۷۴	زمان استنتاج مدل در مسئله تشخیص ارقام	۹-۵
۷۴	گراف نظارت بر پردازنده گرافیکی در مسئله تشخیص ارقام (Grafana (داشبورد	۱۰-۵
۷۴	مجموعه داده احساسات در بازار سهام	۱۱-۵
۷۵	خط لوله آموزش مسئله تحلیل احساسات در بازار سهام	۱۲-۵
۷۶	چرخه کاری آموزش مسئله تحلیل احساسات در بازار سهام	۱۳-۵
۷۷	خط لوله استقرار مسئله تحلیل احساسات در بازار سهام	۱۴-۵
۷۸	گراف استنتاج مدل در مسئله تحلیل احساسات در بازار سهام (Grafana (داشبورد	۱۵-۵
۷۹	زمان استنتاج مدل در مسئله تحلیل احساسات در بازار سهام	۱۶-۵

فهرست کلمات اختصاری

DevOps	Development and Operation
MLOps	Machine Learning Operation
CI	Continuous Integration
CD	Continuous Deployment
CT	Continuous Training
VM	Virtual Machine
IaC	Infrastructure as Code
ETL	Extract, Transform, Load
PVC	Persistent Volume Claim
RBAC	Role-Based Access Control
CNN	Convolutional Neural Network
LSTM	Long Short-Term Memory
DSL	Domain Specific Language
CRD	Custom Resource Definition

فصل اول

مقدمه

۱-۱ تعریف مسئله و بیان چالش‌ها

در عصر حاضر، هوش مصنوعی و یادگیری ماشین به یکی از حیاتی‌ترین و پرکاربردترین فناوری‌ها در صنایع مختلف تبدیل شده‌اند. این فناوری‌ها به شرکت‌ها و سازمان‌ها این امکان را می‌دهند تا بهینه‌سازی فرآیندها، پیش‌بینی روندها و کشف الگوهای پنهان در داده‌ها را با دقت و سرعت بالا انجام دهند. با این حال، بهره‌گیری کامل از قابلیت‌های هوش مصنوعی و یادگیری ماشین نیازمند پیاده‌سازی مؤثر و کارآمد مدل‌های یادگیری ماشین در محیط‌های تولیدی است. در این راستا، مفهومی به نام MLOps^۱ پدید آمده است که به مدیریت، نظارت، و بهروزرسانی مدل‌های یادگیری ماشین در محیط‌های تولیدی اختصاص دارد.

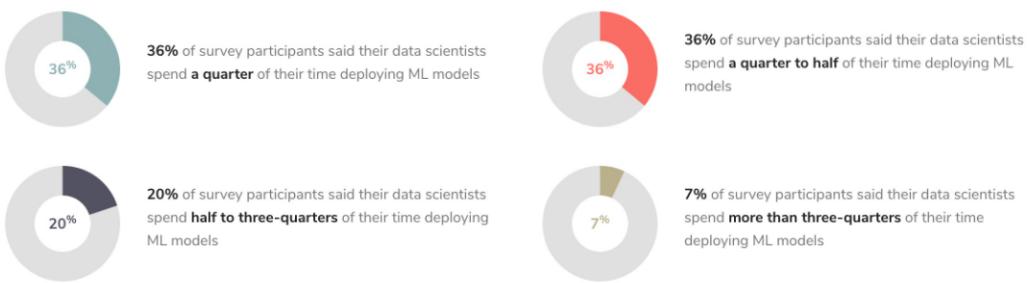
MLOps در واقع ترکیبی از مفاهیم DevOps^۲ و ML^۳ است و به هدف ایجاد یک چارچوب یکپارچه برای توسعه، استقرار و مدیریت مدل‌های یادگیری ماشین به کار می‌رود. این مفهوم به شرکت‌ها کمک می‌کند تا چرخه عمر مدل‌های یادگیری ماشین را از مرحله‌ی توسعه تا مرحله‌ی استقرار و نگهداری به صورت مؤثرتری مدیریت کنند. اهمیت MLOps به دلیل پیچیدگی‌ها و چالش‌های موجود در پیاده‌سازی مدل‌های یادگیری ماشین در محیط‌های واقعی روز به روز بیشتر می‌شود.

به کارگیری مدل‌های یادگیری ماشین در شرکت‌ها به طور فزاینده‌ای رو به افزایش است. با این حال، تنها ساختن یک مدل کافی نیست و برای بهره‌برداری کامل از این مدل‌ها، باید آن‌ها را در محیط واقعی به کار برد. به عبارت دیگر، مدل‌های آموزش دیده‌ی یادگیری ماشین باید به صورت عملیاتی در سیستم‌های نرم‌افزاری اصلی شرکت ادغام شوند. این فرآیند که به عنوان استقرار مدل یادگیری ماشین شناخته می‌شود، به سیستم‌های دیگر اجازه می‌دهد تا داده‌ها را به مدل‌ها ارائه کرده و پیش‌بینی‌ها را دریافت کنند که این پیش‌بینی‌ها سپس به سیستم‌های نرم‌افزاری بازنگرданده می‌شوند.

Machine Learning Operations^۱

Development and Operations^۲

Machine Learning^۳



شکل ۱-۱: زمان استقرار مدل توسط دانشمندان داده

با این وجود، در گزارش [۳] نشان می‌دهد که بسیاری از شرکت‌ها هنوز راه حل مناسبی برای دستیابی به اهداف هوش مصنوعی خود پیدا نکرده‌اند و کاهش فاصله بین ساخت مدل‌های یادگیری ماشین و استقرار عملی آن‌ها همچنان یک چالش اساسی است. ساخت مدل در محیط‌های آزمایشی مانند Jupyter Notebook با استقرار مدل در سیستم‌های عملیاتی که ارزش تجاری ایجاد می‌کنند، تفاوت بنیادی دارد. طبق این گزارش، اگرچه بودجه‌های مربوط به هوش مصنوعی در حال افزایش است، تنها ۲۲ درصد از شرکت‌هایی که از یادگیری ماشین استفاده می‌کنند، موفق به استقرار مدل در محیط عملیاتی شده‌اند. این آمار نشان‌دهنده‌ی یک فاصله‌ی بزرگ در فرآیند پیاده‌سازی می‌باشد.

علاوه بر این، در گزارش [۳] به بررسی وضعیت یادگیری ماشین در سازمان‌ها پرداخته و از حدود ۷۵۰ نفر شامل کارشناسان، مدیران پژوهه و مدیران اجرایی نظرسنجی راجع به پیاده‌سازی مدل‌های یادگیری ماشین انجام داده است. در این نظرسنجی، نیمی از پاسخ‌دهندگان اظهار داشتند که پیاده‌سازی مدل‌های یادگیری ماشین در شرکت‌هاییشان بین یک هفته تا سه ماه طول می‌کشد، در حالی که حدود ۱۸ درصد این مدت را بین سه ماه تا یک سال برآورد کردند (شکل ۱-۱).

پیاده‌سازی مدل‌های یادگیری ماشین در محیط‌های تولیدی با چالش‌های متعددی همراه است. از جمله‌ی این چالش‌ها می‌توان به مدیریت نسخه‌های مختلف مدل‌ها، نظارت بر عملکرد مدل‌ها، اطمینان از تکرارپذیری و قابلیت بازتولید نتایج، و همچنین سازگاری با تغییرات داده‌ها و نیازمندی‌های کسب‌وکار اشاره کرد. MLOps با ارائه‌ی راهکارهای ساختاریافته و ابزارهای مناسب، به کاهش این چالش‌ها کمک می‌کند و فرآیند استقرار و مدیریت مدل‌های یادگیری ماشین را بهبود می‌بخشد.

در انتها باید اضافه کرد که یکی از دلایل اصلی برای توسعه و استفاده از پلتفرم بومی MLOps در ایران، تحریم‌های اعمال شده توسط سرویس‌های بزرگ بین‌المللی مانند Microsoft Azure و Amazon SageMaker AI، Google Vertex AI و ML است. این تحریم‌ها دسترسی کاربران ایرانی به این سرویس‌ها را محدود کرده و نیاز به یک راه حل بومی که بدون وابستگی به سرویس‌های خارجی کار کند را افزایش داده است. با توسعه‌ی یک پلتفرم بومی، شرکت‌ها و سازمان‌های ایرانی قادر خواهند بود بدون نگرانی از تحریم‌ها و محدودیت‌ها، به توسعه و استقرار مدل‌های یادگیری ماشین بپردازنند. علاوه بر این، این

ارائه‌دهندگان ابری غالباً راه حل قابل قبولی برای شرکت‌هایی که روی سیستم‌های نرم‌افزاری نظارتی یا نرم‌افزارهایی با نگرانی‌های مربوط به حریم خصوصی کاربر کار می‌کنند، نیستند. آن‌ها به راه حل‌هایی نیاز دارند که قابل اجرا روی ابرها و دستگاه‌های داخلی باشد.

تحقیق حاضر با هدف طراحی و پیاده‌سازی یک پلتفرم MLOps به منظور رفع نیازهای موجود در مدیریت مدل‌های یادگیری ماشین و بهبود فرآیندهای توسعه و استقرار آن‌ها انجام شده است. این پلتفرم به گونه‌ای طراحی شده است که قابلیت‌های مختلفی نظیر مدیریت چرخه عمر مدل، نظارت بر عملکرد، بهروزرسانی خودکار، و تعامل آسان با ابزارهای موجود را فراهم آورد. علاوه بر این، پیاده‌سازی این پلتفرم به صورت ابری انجام می‌شود تا انعطاف‌پذیری و مقیاس‌پذیری بیشتری را به کاربران ارائه دهد.

۲-۱ روش تحقیق

این پایان‌نامه به بررسی و توسعه‌ی یک پلتفرم MLOps می‌پردازد. روش تحقیق به گونه‌ای طراحی شده است که پاسخگوی سوالات تحقیقاتی زیر باشد:

۱. چه نیازهایی برای یک پلتفرم MLOps مدرن وجود دارد؟
۲. چقدر امکان‌پذیر است که پلتفرم MLOps با استفاده از ابزارهای متن‌باز، به صورت ابری طراحی و پیاده‌سازی شود؟
۳. راه حل تا چه حد می‌تواند از ارائه‌دهندگان ابر مستقل باشد؟

ابتدا، مسئله به دقت شناسایی و سه سوال تحقیقاتی اصلی مطرح می‌شوند. سپس، مرور جامعی از منابع علمی و مراجع و اسناد اینترنتی برای شناسایی بهترین روش‌ها، ابزارها و چالش‌های موجود در پیاده‌سازی پلتفرم‌های MLOps انجام می‌شود. براساس نتایج به دست آمده از این مرور ادبیات، پلتفرمی با استفاده از ابزارهای متن‌باز ابری طراحی و توسعه داده می‌شود که سهولت استفاده، کارآمدی و قابلیت توسعه را تضمین می‌کند.

برای طراحی و پیاده‌سازی یک پلتفرم MLOps به صورت ابری، ابتدا باید نیازمندی‌ها و اهداف پروژه مشخص شوند. سپس، ابزارها و تکنولوژی‌های مناسب انتخاب و معماری سیستم طراحی می‌شود. در نهایت، فرآیندهای توسعه، استقرار و نظارت بر مدل‌ها به صورت خودکارسازی شده پیاده‌سازی می‌شوند.

ابزار متن باز

برخی از ابزارها و تکنولوژی‌های کلیدی متن باز که برای پیاده‌سازی پلتفرم از آن‌ها استفاده شده است، عبارتند از:

- Kubernetes: برای مدیریت کانتینرها و فرآیندهای توزیع شده استفاده می‌شود.
- Kubeflow: یک پلتفرم منبع باز برای اجرای جریان کاری^۴ یادگیری ماشین که بر روی کوبرتیز اجرا می‌شود.
- Jenkins: یک ابزار اتوماسیون برای پیاده‌سازی فرآیندهای CI/CD که امکان خودکارسازی توسعه و استقرار نرم‌افزار را فراهم می‌کند.
- Prometheus: یک سیستم نظارتی و مانیتورینگ که برای جمع‌آوری و تحلیل داده‌های عملکردی استفاده می‌شود.
- MinIO: یک سیستم ذخیره‌سازی شبیه‌گرایی با سازگاری بالا و مقیاس‌پذیری برای مدیریت داده‌های بزرگ طراحی شده است.

طراحی معماری سیستم

معماری یک پلتفرم MLOps شامل لایه‌های مختلفی است که هر کدام نقش خاصی در فرآیند توسعه و استقرار مدل‌ها ایفا می‌کنند. این لایه‌ها عبارتند از:

- لایه‌ی داده: شامل ابزارها و فرآیندهای جمع‌آوری، پردازش و ذخیره‌سازی داده‌ها می‌باشد. این لایه اطمینان حاصل می‌کند که داده‌های مورد نیاز برای آموزش و ارزیابی مدل‌ها به صورت بهینه و قابل‌دسترس هستند.
- لایه‌ی مدل‌سازی: شامل ابزارها و محیط‌های توسعه، آموزش و ارزیابی مدل‌ها. این لایه به دانشمندان داده و مهندسان یادگیری ماشین امکان می‌دهد تا مدل‌های خود را به سرعت و با دقت بالا توسعه دهند و ارزیابی کنند.
- لایه‌ی استقرار: شامل ابزارها و فرآیندهای استقرار و اجرای مدل‌ها در محیط تولید می‌باشد. این لایه تضمین می‌کند که مدل‌ها به صورت پایدار و قابل‌اعتماد در محیط تولید اجرا شوند.
- لایه‌ی نظارت: شامل ابزارها و فرآیندهای نظارتی و مانیتورینگ برای عملکرد مدل‌ها پس از استقرار می‌باشد. این لایه به تیم‌های عملیاتی امکان می‌دهد تا عملکرد مدل‌ها را به صورت مداوم نظارت کنند و در صورت نیاز به روزرسانی‌ها و تغییرات لازم را اعمال کنند.

⁴ Workflow

پیاده‌سازی فرآیندهای CI/CD

پیاده‌سازی فرآیندهای CI/CD برای مدل‌های یادگیری ماشین به معنای ایجاد یک چرخهٔ تولید و تحویل مدام است که از

توسعهٔ تا نظارت بر مدل‌ها را پوشش می‌دهد. این فرآیند شامل مراحل زیر است:

- توسعه و ادغام: کدها و تغییرات جدید به مخزن کد ادغام می‌شوند و تست‌های اتوماتیک اجرا می‌گردند. این مرحله

شامل اجرای تست‌های واحد، تست‌های عملکردی برای اطمینان از صحت و عملکرد کد جدید

است.

- آموزش و ارزیابی: مدل‌ها با استفاده از داده‌های جدید آموزش داده شده و ارزیابی می‌شوند. این مرحله شامل فرآیندهای

انتخاب مدل، تنظیم هایپرپارامترها و ارزیابی مدل‌ها برای اطمینان از عملکرد بهینه‌ی آنها است.

- استقرار: مدل‌های تایید شده به محیط تولید منتقل می‌شوند. این مرحله شامل فرآیندهای بسته‌بندی مدل‌ها، پیکربندی

محیط تولید و استقرار مدل‌ها به صورت خودکار است.

- نظارت و بهروزرسانی: عملکرد مدل‌ها نظارت می‌شود و در صورت نیاز به روزرسانی‌های لازم اعمال می‌گردد. این

مرحله شامل جمع‌آوری داده‌های عملکردی، شناسایی انحرافات و اعمال تغییرات و بهروزرسانی‌های لازم برای بهبود

عملکرد مدل‌ها است.

این پلتفرم با استفاده از یک پروژهٔ نمونهٔ یادگیری ماشین پیاده‌سازی و آزمایش می‌شود. نتایج حاصل از این آزمایش‌ها ارزیابی و تحلیل می‌شوند تا میزان پاسخگویی به نیازهای مدرن، امکان‌پذیری پیاده‌سازی با ابزارهای متن‌باز ابری، و میزان استقلال از ارائه‌دهنگان ابر مشخص شود. در نهایت، یافته‌های تحقیق به صورت خلاصه ارائه شده و پیشنهاداتی برای تحقیقات آینده و بهبود پلتفرم ارائه می‌گردد.

۳-۱ ساختار پایان نامه

در این پایان‌نامه، ساختار به شش بخش اصلی تقسیم می‌شود. در فصل اول، به معرفی و بیان مسائل و چالش‌های تحقیق پرداخته می‌شود. فصل دوم و سوم به بررسی مفاهیم کلیدی DevOps و MLOps، همراه با مرور پیشینه‌ی موضوع و تکنولوژی‌های مرتبط، می‌پردازد. ابزارها و زیرساخت‌های مهم مانند Docker و Kubernetes به تفصیل معرفی می‌شوند. فصل چهارم به طراحی پلتفرم و سیستم‌های مدیریت مرتبط با MLOps اختصاص دارد، که براساس اصول و اجزای معرفی شده

در فصل‌های قبل، یک پلتفرم MLOps طراحی می‌گردد. همچنین، ابزارهای مختلف بررسی و مناسب‌ترین آن‌ها برای طراحی انتخاب می‌شوند. فصل پنجم به پیاده‌سازی معماری طراحی شده اختصاص دارد و نتایج حاصل از پیاده‌سازی پلتفرم به نمایش گذشته می‌شود. برای اعتبارسنجی، دو مسئله‌ی یادگیری ماشین بر روی این پلتفرم اجرا می‌شود. نهایتاً، فصل ششم شامل جمع‌بندی، پیشنهادات و نتیجه‌گیری کلی از تحقیق است.

فصل ۲

مرور مفاهیم پایه

۱-۲ مقدمه

در این بخش از پایاننامه، به بررسی متداول‌وزی DevOps و نقش کلیدی آن در بهبود فرآیندهای توسعه و عملیات نرم‌افزار پرداخته می‌شود. ابتدا مفاهیم پایه DevOps و اهمیت آن در ایجاد همکاری مؤثر بین تیم‌های توسعه و عملیات معرفی می‌شوند. سپس به چرخه‌ی کاری DevOps و نحوه استفاده از خط لوله‌ی CI/CD برای خودکارسازی فرآیندها و افزایش سرعت و کارایی می‌پردازیم. همچنین مزایای این متداول‌وزی، از جمله افزایش سرعت، کاهش هزینه‌ها و بهبود پایداری و قابلیت اطمینان سیستم‌ها، مورد بحث قرار می‌گیرد. در ادامه، فناوری‌های مجازی‌سازی و کانتیرسازی به عنوان ابزارهایی برای بهینه‌سازی استفاده از منابع و مدیریت آسان‌تر محیط‌های توسعه و تولید بررسی می‌شوند. همچنین، نقش کوبرنتیز در هماهنگ‌سازی و مدیریت کانتیرها و اهمیت آن در تسهیل استقرار و مقیاس‌پذیری برنامه‌ها تحلیل می‌شود. این مباحث به عنوان زیربنای مهمی برای درک کامل DevOps، MLOps و ابزارهای مرتبط با آن، ارائه می‌شوند.

۲-۲ DevOps

۱-۲-۲ تعریف

دواپس که از اتحاد واژگان Development و Operation به وجود آمده است؛ ترکیبی از ابزارها، کنش‌ها و فرهنگ کاری است که تیم‌های توسعه^۱ و عملیات^۲ را به همکاری مؤثرتر نزدیک می‌کند و کسب‌وکارها با استفاده از آن می‌توانند اپلیکیشن‌ها و سرویس‌هایشان را با سرعت بالاتری نسبت به روش‌های سنتی تحويل دهند. همین سریع‌تر شدن سرعت توسعه و انتشار

Development^۱
Operation^۲

نرم افزار، سازمان ها را قادر می سازد تا در مقایسه با کسب و کارهایی که هنوز از روش های سنتی توسعه نرم افزار استفاده می کنند، خدمات بهتری به مشتریانشان ارائه دهند. در واقع، دوپس سعی دارد تا مشکل جدایی تیم های مختلف را رفع کرده و یک فرهنگ سازمانی یکپارچه را میان تیم های مختلفی که در حال توسعه یک نرم افزار هستند ایجاد کند. از این جهت بسیاری از کارها می تواند به صورت خودکار پیش رفته و در نهایت همه چیز با سرعت بیشتری صورت بگیرد [۴، ۵]. این خودکارسازی با استفاده از خط لوله CI/CD از منبع کد شروع می شود و تا مانیتورینگ محصول ادامه می یابد [۶].

تا قبل از تشکیل دوپس، تیم های توسعه نرم افزار و تیم های عملیاتی در محیط های جداگانه کار می کردند. هدف تیم توسعه، تولید محصول جدید و یا افزودن ویژگی های جدیدی روی محصولات قبلی بود. هدف تیم عملیاتی نیز ثابت نگه داشتن وضعیت موجود سرویس ها برای پایداری بیشتر بود. به مرور زمان، در فرآیند توسعه نرم افزار، روش های چاپک^۳ ایجاد شد تا با مشتری تعامل بهتری برقرار شود و نیازهایی که دارد به محصول اضافه شود [۷]. جدایی دو تیم توسعه و عملیات از هم باعث می شد که در فرآیند تولید محصول و استقرار^۴ آن، اتلاف وقت ایجاد شود و محصول دیرتر به دست مشتری برسد [۸].

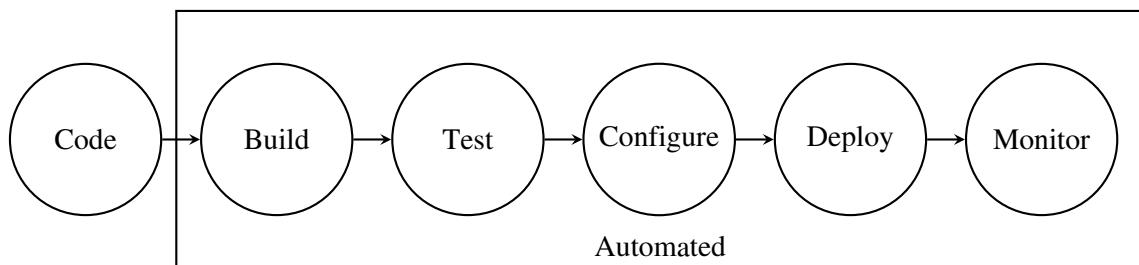
۲-۲ چرخه کاری

همانطور که در شکل ۱-۲ مشاهده می کنید، DevOps قصد دارد از ابزار و جریان های کاری^۵ برای خودکارسازی یک یا چند مورد از موارد زیر استفاده کند:

۱. کدنویسی: شامل توسعه، بازبینی کد و ابزارهای کنترل نسخه است. مثلا، یک تیم تصمیم می گیرد از گیت^۶ به عنوان ابزار کنترل نسخه و از گیت هاب^۷ نیز به عنوان یک مخزن راه دور استفاده کند. این تیم مجموعه ای از دستورالعمل های سبک کدنویسی را با استفاده از ابزاری نظیر Linter به همراه حداقل درصد پوشش تست تعریف کرده و با تعیین استراتژی انشعباب مبتنی بر تنه^۸ تغییرات خود را به منظور بازبینی برای ادغام با انشعباب اصلی^۹ برای توسعه دهنده ارشد ارسال می کند [۹].

۲. ساخت: شامل ایجاد و ذخیره خودکار مولفه^{۱۰} ها می باشد. به طور مثال، یک تیم تصمیم می گیرد یک Container image قابل اجرا از محصول خود ایجاد کند.

Agile ^۳
Deploy ^۴
Workflow ^۵
Git ^۶
Github ^۷
Trunk-Based ^۸
Merge request ^۹
Artifact ^{۱۰}



شکل ۱-۲: مراحل DevOps

۳. تست: شامل ابزارهایی برای تست محصول می‌باشد. تیم محیطی را به منظور تست هر تغییر جدید راهاندازی می‌کند که در آن مجموعه‌ای از آزمایش‌ها مانند آزمون واحد^{۱۱}، آزمون یکپارچگی^{۱۲} و ... به طور خودکار در برایر هر ویرایش کد اجرا می‌شود. ادغام و تست کد به طور مکرر، به تیم‌های توسعه کمک می‌کند تا از کیفیت کدشان اطمینان حاصل کرده و جلوی خطاهای احتمالی را بگیرند.

۴. پیکربندی: شامل پیکربندی و مدیریت خودکار زیرساخت می‌باشد. این مورد شامل مجموعه‌ای از اسکریپت‌هایی برای بازنولید محیط در حال اجرا و زیرساخت نرمافزاری شامل سیستم‌عامل تا پایگاه داده و سرویس‌های خاص و پیکربندی شبکه آن‌ها می‌باشد [۱۰، ۱۱].

۵. استقرار: این مرحله شامل استراتژی استقرار است. به طور مثال، تیم می‌تواند تصمیم بگیرد که یک محصول به طور مستقیم منتشر شود یا ابتدا در یک محیط آزمایشی مورد ارزیابی قرار گیرد. همچنین، در موقعی که مشکلی در استقرار وجود دارد، چه کاری انجام دهند و استراتژی بازگشت^{۱۳} خود را پیاده‌سازی کنند.

۶. نظارت: از عملکرد محصول تا نظارت بر تجربه کاربر نهایی را شامل می‌شود. به عنوان مثال، می‌تواند مدت زمان درخواست‌های پایگاه داده یا بارگذاری وب‌سایت یا تعداد کاربرانی که از ویژگی‌های خاص محصول استفاده می‌کنند یا تعداد بازدیدکنندگان از یک وب‌سایت که به ثبت‌نام ختم می‌شود یا تعداد کاربران جدید در یک مجموعه زمانی خاص را پوشش دهد. مرحله نظارت همچنین شامل هشدار خودکار خرابی‌ها نیز می‌باشد (به عنوان مثال، آستانه استفاده از CPU [۱۲]). در نهایت، نظارت بر محیط تولید به منظور اطمینان از صحت کارکرد صحیح محصول ضروری است.

Unit test^{۱۱}
Integration test^{۱۲}
Rollback^{۱۳}

خط لوله CI/CD ۳-۲-۲

در دنیای توسعه نرم افزار دستیابی به بهره‌وری بالا، کیفیت مطلوب محصول و رضایت مشتری از اهداف اصلی هر سازمانی است. متداول‌ترین DevOps و رویکردهای مرتبط با آن مانند ادغام مداوم^{۱۴}، تحویل مداوم^{۱۵} و استقرار مداوم^{۱۶} به طور فزاینده‌ای محبوب شده‌اند زیرا به سازمان‌ها کمک می‌کنند تا با سرعت و کارآمدی بیشتری به این اهداف دست یابند [۸].

ادغام مداوم به فرآیندی اطلاق می‌شود که در آن توسعه‌دهنگان برنامه‌های خود را به طور مداوم (معمولًاً چندین بار در روز) در یک مخزن مشترک ادغام می‌کنند. به محض ادغام کد، یک سری از تست‌های خودکار اجرا می‌شود تا اطمینان حاصل شود که این تغییرات جدید باعث بروز مشکل در نرم افزار نشده‌اند [۱۳]. این تست‌ها شامل تست‌های واحد، تست‌های یکپارچگی و تست‌های کارکردی می‌باشند. از آنجایی که برنامه‌های کاربردی پیشرفته کنونی در چندین پلتفرم و ابزار مختلف توسعه می‌یابند، نیاز به مکانیزمی برای ادغام و تایید تغییرات مختلف، اهمیت بالاتری پیدا می‌کند.

تحویل مداوم ادامه‌ای بر ادغام مداوم است و به تیم‌ها این امکان را می‌دهد تا نرم افزار را پس از هر تغییر مهم در کد به مرحله تولید برسانند. در این مدل، هر خروجی که از فرایند CI عبور کرده و تست‌های لازم را با موفقیت پشت سر گذاشته باشد، به صورت خودکار آماده انتشار می‌شود [۱۳]. به عبارتی دیگر، هدف از تحویل مداوم، داشتن پایگاه کدی است که همیشه آماده استقرار در محیط تولید باشد. این فرایند ممکن است شامل تست‌های اضافی برای ارزیابی عملکرد، امنیت و سازگاری با محیط‌های تولید نیز باشد.

استقرار مداوم که گاهی با تحویل مداوم اشتباه گرفته می‌شود، به فرآیندی اطلاق می‌شود که در آن هر تغییر در کد که تمام مراحل تست و تأیید را با موفقیت پشت سر می‌گذارد، به صورت خودکار در محیط تولید قرار می‌گیرد [۱۳]. این به معنای آن است که نسخه‌های جدید نرم افزار می‌توانند به طور مداوم و بدون دخالت دستی به کاربران نهایی تحویل داده شوند. این رویکرد به تیم‌ها کمک می‌کند تا سریع‌تر به بازخوردها پاسخ دهند و بهبودهای مستمری را در محصول خود اعمال کنند، اما نیازمند یک فرآیند آزمایشی بسیار قوی و اطمینان از کیفیت کد است.

فرآیند کامل CI/CD که در شکل ۱-۲ هم به عنوان بخشی از چرخه کاری توضیح داده شد [۸]، با یک فرآیند ساخت شروع می‌شود. در این مرحله کد توسط ابزارهای مرتبط که در جدول ۱-۲ ذکر شده است، تبدیل به نرم افزار قابل اجرا می‌شود. پس از این مرحله، تست‌های خودکار که شامل تست‌های واحد، تست‌های یکپارچه‌سازی و تست‌های رابط کاربری هستند، اجرا می‌شوند تا اطمینان حاصل شود که تغییرات جدید باعث بروز خطا در نرم افزار نمی‌شوند. در صورت موفقیت آمیز بودن

Continuous Integration (CI)^{۱۴}

Continuous Delivery (CD)^{۱۵}

Continuous Deployment (CD)^{۱۶}

جدول ۲-۱: نمونه‌هایی از ابزار برای مراحل خاص اتوماسیون خط لوله CI/CD

Phase	Tools
Build	Gradle, Bazel, Docker
Test	Selenium, pytest
Configure	Ansible, Terraform
Deploy	ArgoCD, Jenkins
Monitor	Prometheus, Sentry

تست‌ها، یک نسخه قابل اجرا از کد نسخه‌گذاری شده و در مخازنی همانند Nexus نگهداری می‌شود.

در مرحله پیکربندی، تنظیم محیط لازم برای نصب و استفاده از نرم‌افزار انجام می‌شود. دورویکرد اصلی برای این کار وجود دارد: مرحله به مرحله^{۱۷} و اعلامی^{۱۸}. در رویکرد اول، پیش‌نیازها به ترتیب آماده‌سازی می‌شوند و شکست در هر مرحله می‌تواند به عدم انجام دادن مراحل بعدی منجر شود. این رویکرد، که اغلب با استفاده از ابزارهایی مانند Ansible پیاده‌سازی می‌شود، زمانی مفید است که نیاز به اعمال تغییرات جزئی بر محیط باشد. در مقابل، رویکرد اعلامی به طور همزمان کل محیط را بر اساس یک حالت نهایی تعریف‌شده آماده می‌کند. این رویکرد باعث می‌شود که در صورت بروز خطا در یک بخش، سایر بخش‌ها تحت تأثیر قرار نگیرند. برای اجرای این رویکرد، می‌توان از ابزارهایی مثل Terraform استفاده کرد [۱۴].

پس از این، مرحله‌ی استقرار آغاز می‌شود که در آن نرم‌افزار به محیط‌های تست، توسعه یا تولید منتقل می‌شود. این فرآیند اغلب شامل مکانیزم‌هایی برای پشتیبانی و بازگرداندن نسخه‌های قبلی در صورت بروز مشکل است. یکی از قسمت‌های فرآیند کامل این خط لوله نیز با استفاده از ابزاری مانند CircleCI، Jenkins و Gitlab است.

در آخر نیز مرحله نظارت انجام می‌شود. ابزارهای نظارتی نظیر Grafana و Prometheus معمولاً شامل نمودارها، گزارش‌ها و آمارهایی هستند که به شما اطلاعاتی در مورد وضعیت فعلی خط لوله و عملکرد برنامه‌های آزمایشی و انتشارات را ارائه می‌دهند. همچنین اطلاعاتی مانند زمان طول کشیده برای هر مرحله، تعداد خطاهای متعدد زمان بین خرابی‌ها^{۱۹} از جمله آمارهایی هستند که ممکن است در این مرحله نمایش داده شوند. لازم به ذکر است که می‌توان به منظور بررسی سبک کدنویسی و اعمال استانداردهای تیم توسعه در ابتدا خط لوله بخشی را قرار داد تا از استاندارد بودن کد اطمینان یابد. در این بخش می‌توان از Linter‌ها یا pre-commit hooks استفاده کرد. معروف‌ترین ابزار برای هر مرحله را در جدول ۲-۲ نیز مشاهده می‌کنید.

به هنگام طراحی و پیاده‌سازی یک خط لوله CI/CD باید به نکات زیر توجه کرد [۸، ۱۴، ۱۳]:

• قابلیت بازگشت به حالت قبل^{۲۰}

Procedural^{۱۷}

Declarative^{۱۸}

Mean time between failures (MTBF)^{۱۹}

Rollback^{۲۰}

- قابلیت مشاهده^{۲۱} و هشدار دادن^{۲۲}

- امنیت

- مدت زمان اجرای خط لوله

برای هر نسخه از کد باید یک استراتژی بازگشت وجود داشته باشد تا اگر مشکلی پیش آمد، به نسخه قبلی بازگردانده شود. یک راه حل آسان برای بازگشت می‌تواند اجرای نسخه قدیمی‌تر از طریق همان خط لوله CI/CD باشد. بازگشت به ورژن قبلی همیشه ساده نیست، چراکه اگر سرویس در حال اجرا قابل بازگشت باشد، باید به بازگرداندن داده‌ها قبلی و زمان توقف هنگام استقرار نسخه جدید نیز توجه کرد.

هر انتشار باید شفاف باشد که چه تغییراتی اعمال شده و چه کسی تغییرات را تأیید کرده است. همچنین تیم توسعه باید بداند که استقرار موفقیت‌آمیز بوده و چه زمانی انجام شده است. در نهایت، باید هشدارهای واضحی از کد خراب در خط لوله و خطای احتمالی در انتشار وجود داشته باشد. اگر مشکلی در استقرار پیش آید و هیچ سابقه واضحی از تغییرات وجود نداشته باشد، بازگشت به حالت قبل دشوار خواهد بود. علاوه بر این، دلیل بروز این مشکل در استقرار نیز برای تیم نامعلوم است. تصور کنید که یک توسعه‌دهنده به صورت دستی به یک ماشین محیط تولید دسترسی پیدا کرده و به طور تصادفی یک فایل کلیدی را حذف می‌کند که پس از چند روز باعث خرابی‌های سیستم می‌شود. هیچ ردی از این تغییرات وجود ندارد، هیچ‌کس نمی‌داند کجا را باید به حالت قبلی برگرداند و حتی بازگشت به کد قبلی ممکن است کمکی نکند زیرا فایل گمشده ممکن است در ورژن قبلی بازتولید نشود.

توجه به امنیت در فرآیندهای CI/CD بسیار حیاتی است تا سلامت و امنیت فرآیندهای توسعه و ارسال نرم‌افزار حفظ شود. اجرای کنترل دسترسی بر اساس نقش^{۲۳} ضروری است تا فقط افراد مجاز بتوانند تغییراتی در فرآیند CI/CD اعمال کرده و کد را ارسال کنند. این شامل کنترل دسترسی به ابزارهای CI/CD نظیر Jenkins و همچنین به هر سیستم مرکز مانند مخازن کد منبع است. همچنین مدیریت اسرار^{۲۴} جنبه اساسی امنیت خط لوله است. اسراری مانند کلیدهای API، رمزعبورها و گواهی‌نامه‌ها باید به طور امن ذخیره و دسترسی‌پذیر باشند. استفاده از ابزارهایی مانند HashiCorp Vault می‌تواند به مدیریت امنیت اسرار کمک کند.

مدت زمان اجرای کامل یک خط لوله از ساخت تا استقرار برای حفظ چابکی بسیار مهم است. تست‌ها و ساخت‌های طولانی مدت می‌توانند منجر به تداخل با خط لوله‌های دیگر شود. بهبود و بهینه‌سازی این فرآیند از طریق اجرای موازی

Observability^{۲۱}

Alerting^{۲۲}

Role-Based Access Control (RBAC)^{۲۳}

Secrets^{۲۴}

تست‌ها، بهبود قابلیت مقیاس‌پذیری زیرساخت و بهینه‌سازی کد می‌تواند به کاهش زمان مورد نیاز برای تست و ساخت و بهبود جریان کار توسعه کمک کند.

در محصولاتی که از یادگیری ماشین استفاده می‌کنند نیز مراحل خط لوله برای رسیدگی به چرخه عمر مدل و داده‌ها افزایش می‌یابد، اما عناصر، مزایا و اهداف یکسان هستند.

۴-۲-۲ مزایای متداول‌تری DevOps

این متداول‌تری یک رویکرد نوآورانه در توسعه نرم‌افزار و عملیات است که مزایای بسیاری برای بهبود عملکرد سازمانی^{۲۵} ارائه می‌دهد [۱۵]. ادغام این روش‌ها می‌تواند نحوه مواجهه تیم‌ها با چالش‌های پروژه و تعامل با فناوری را تغییر داده و منجر به افزایش کارایی، قابلیت اطمینان و رضایت شود [۵].

۱. افزایش سرعت و کارایی: با خودکارسازی فرایند انتشار نرم‌افزار از طریق CI/CD، تیم‌ها می‌توانند فرکانس و سرعت انتشارها را افزایش داده که منجر به افزایش سرعت پاسخ‌دهی به مشتری شده و مزیت رقابتی ایجاد می‌کند [۶].

۲. ایجاد محیط‌های عملیاتی پایدارتر: تضمین قابلیت اطمینان بهروزرسانی‌های برنامه و تغییرات زیرساخت یکی از مزایای مهم این متداول‌تری می‌باشد. از طریق خط لوله CI/CD، هر تغییری برای اطمینان از کارایی و اینمی ادغام با محیط تولید آزمایش می‌شود تا از انتشار نسخه‌های معیوب جلوگیری شود. یکی از شاخص‌های اصلی پایداری، انتشارهای متناوب و مکرر است. با استفاده از این متداول‌تری توسعه‌دهندگان می‌توانند خطاهای را سریع‌تر شناسایی و رفع کنند. این موضوع باعث کاهش شاخص MTTR^{۲۶} می‌شود. این شاخص مدت زمان برگشت به وضعیت پایدار بعد از وقوع خطای اشکال را نشان می‌دهد و هرچه مقدار آن کمتر باشد، پایداری سیستم بیشتر است [۸]. علاوه بر انتشار پیوسته و مستمر، نرم‌افزارهای مانیتورینگ هم با پایش مداوم نرم‌افزار و سرورها و ایجاد دسترسی به اطلاعات حیاتی نرم‌افزار و محیط عملیاتی برای مهندسان، نقش مهمی در شناسایی و رفع خطاهای در نتیجه حفظ پایداری دارند.

۳. مقیاس‌پذیری: تسهیل کننده مدیریت مقیاس‌پذیر زیرساخت‌ها و فرآیندهای توسعه است. تکنیک‌هایی مانند زیرساخت به عنوان کد^{۲۷} مدیریت محیط‌های توسعه، آزمایش و تولید را به شکلی تکرارپذیر و کارآمد ساده‌سازی می‌کنند [۵].

۴. صرفه‌جویی در هزینه‌ها و منابع: علاوه بر مدیریت بهتر عملکرد و ارتباطات، هزینه‌ها و منابع را هم به نسبت روش‌های قدیمی کاهش می‌دهد. با استفاده از این متداول‌تری و خط لوله CI/CD طول چرخه‌ها کوتاه‌تر و نتایج کمی و کیفی بهتر

^{۲۵} Organization performance

^{۲۶} Mean Time To Recover

^{۲۷} Infrastructure as Code (IaC)

می‌شوند و در نتیجه هزینه‌ها نیز کاهش پیدا می‌کنند. این فرآیند حتی نیاز به منابع سخت‌افزاری و منابع انسانی را هم کاهش می‌دهد. با استفاده از معماری ماژولار، اجزا و منابع به خوبی دسته‌بندی شده و سازمان‌ها می‌توانند به راحتی از فضا و رایانش ابری برای انجام کارها استفاده کنند. چاکری در این متالوژی اهمیت زیادی دارد لذا فناوری ابری نیز این چاکری را به تیم‌ها ارائه و سرعت و هماهنگی بین تیم‌ها را افزایش می‌دهد. با کمک این فناوری، حتی اگر در فرآیند توسعه و عملیات نیاز به منابع جدید و بیشتر بود، با ثبت یک درخواست ساده در عرض چند دقیقه منابع جدید در اختیار سازمان قرار می‌گیرد. از مزایای دیگر استفاده از رایانش ابری می‌توان به حداقل شدن هزینه‌های شروع و عملیاتی پروره، بهبود امنیت، افزایش مشارکت و بهبود دسترسی و کاربری داده‌ها اشاره کرد.

۵. تعزیز ایزوله‌گرایی: در بسیاری از سازمان‌ها، به دلایل امنیتی و مدیریتی، اطلاعات در تیم‌ها به طور جداگانه نگهداری می‌شوند و این باعث ایجاد سیلوهای سازمانی شده که مانع از گردش منظم داده و اطلاعات در سازمان می‌شود. با این حال، با بهره‌گیری از این متالوژی وجود همکاری فعال در تیم‌ها، ارتباطات بهبود می‌یابد. این امر باعث می‌شود که اطلاعات به طور موثرتر جریان یابد، کارایی تیم‌ها افزایش یابد و در نتیجه، کارایی کلی سازمان بهبود پیدا کند.

۳-۲ مجازی سازی و کانتینرها

۱-۳-۲ مجازی سازی

تکنولوژی مجازی سازی^{۲۸} به روشنی اشاره دارد که در آن منابع سخت‌افزاری یک سیستم فیزیکی به چندین محیط مجازی تقسیم می‌شوند. این تکنولوژی به سازمان‌ها این امکان را می‌دهد تا منابع خود را به شیوه‌ای کارآمدتر استفاده کنند، زیرا می‌توانند چندین سیستم عامل و برنامه را روی یک سرور فیزیکی اجرا کنند. مجازی سازی انواع مختلفی دارد، از جمله مجازی سازی سرور، دسکتاپ، نرم‌افزار و شبکه، که هر کدام کاربردهای خاص خود را دارند [۱، ۱۶].

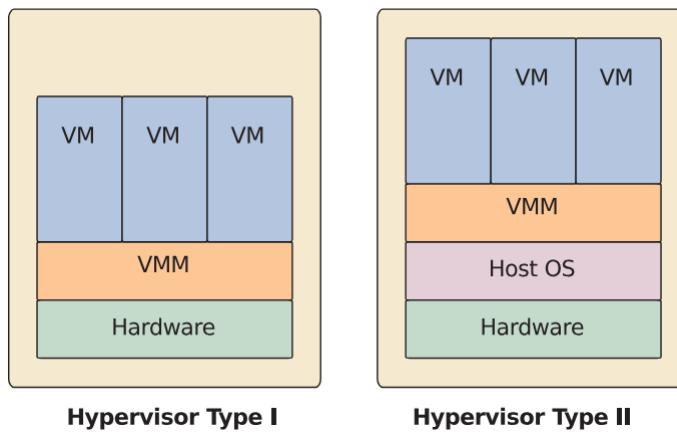
مجازی سازی سرور یکی از تکنولوژی‌های کلیدی در مدیریت و بهره‌برداری از داده‌ها و منابع سخت‌افزاری در مراکز داده است. این فناوری امکان تقسیم یک سرور فیزیکی^{۲۹} به چندین سرور مجازی را می‌دهد، به طوری که هر سرور مجازی می‌تواند به صورت مستقل عمل کرده و سیستم عامل و برنامه‌های کاربردی خود را اجرا کند. مجازی سازی سرور معمولاً شامل سه جزء اصلی است [۱]:

• هایپروایزر^{۳۰}

^{۲۸} Virtualization

^{۲۹} Bare-metal

^{۳۰} Hypervisor



شکل ۲-۲: انواع هایپروایزر [۱]

- ماشین مجازی

- سیستم مدیریت مرکزی

هایپروایزر، که گاهی اوقات به عنوان مدیر ماشین مجازی^{۳۱} شناخته می‌شود، نقش محوری در مجازی‌سازی سرور دارد. این نرمافزار بر روی سخت‌افزار سرور نصب می‌شود و وظیفه آن تقسیم منابع سرور فیزیکی، مانند CPU، حافظه، فضای دیسک و شبکه به چندین ماشین مجازی است. هایپروایزرهای دو دسته تقسیم می‌شوند.

هایپروایزر نوع ۱^{۳۲} مستقیماً بر روی سخت‌افزار نصب می‌شود و به طور مستقل از سیستم عامل فیزیکی عمل می‌کند. می‌توان از هایپروایزرهای نوع ۱ معروف به Microsoft Hyper-V، VMware ESXi و KVM اشاره کرد که برای بهینه‌سازی عملکرد و امنیت طراحی شده‌اند.

هایپروایزر نوع ۲^{۳۳} روی یک سیستم عامل میزبان نصب می‌شود و به عنوان یک برنامه درون سیستم عامل عمل می‌کند. از هایپروایزرهای نوع ۲ نیز می‌توان به Oracle VirtualBox و VMware Workstation اشاره کرد. این هایپروایزرهای اغلب برای تست و توسعه مورد استفاده قرار می‌گیرند. در شکل ۲-۲ ساختار آن را مشاهده می‌کنید.

هایپروایزرهای لحاظ انعطاف‌پذیری و امکان پیکربندی متنوع، قابلیت‌های قدرتمندی را برای مدیریت سرورهای مجازی فراهم می‌کنند. آنها می‌توانند به طور خودکار منابع را بین ماشین‌های مجازی تخصیص دهند و امکاناتی مانند تکثیر^{۳۴} و بازیابی فاجعه^{۳۵} را ارائه دهند.

^{۳۱}Virtual Machine Manager (VMM)

^{۳۲}Bare-metal

^{۳۳}Hosted

^{۳۴}Replication

^{۳۵}Disaster Recovery

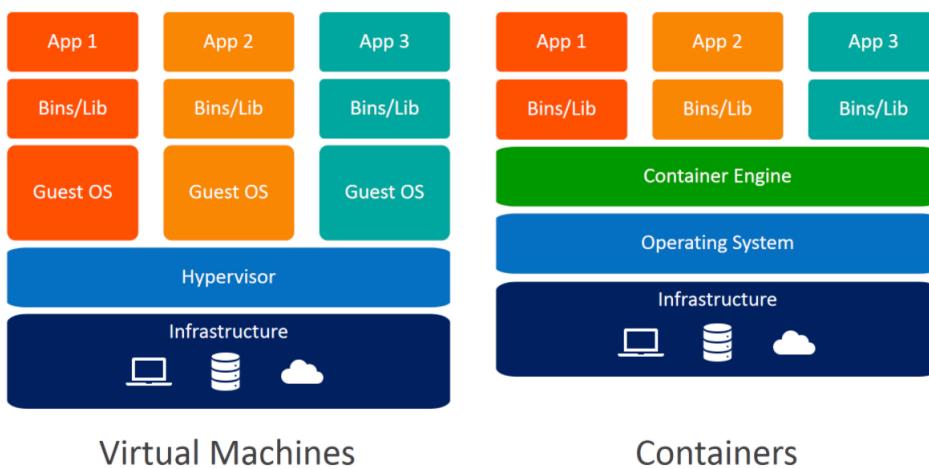
ماشین مجازی^{۳۶} واحدی از منابع مجازی است که شبیه‌سازی یک سرور فیزیکی را انجام می‌دهد. هر VM می‌تواند سیستم عامل خود را داشته باشد و مستقل از دیگر VM‌ها عمل کند. این امر به کاربران اجازه می‌دهد که برنامه‌های متعدد را بدون تداخل با یکدیگر اجرا کنند. VM‌ها از منابع سخت‌افزاری تخصیص داده شده توسط هایپروایزر استفاده می‌کنند و می‌توانند به راحتی از یک سرور فیزیکی به دیگری با استفاده از تکنیک‌هایی نظیر Snapshot منتقل شوند.

استفاده از تکنولوژی مجازی‌سازی نقش بسیار مهمی در فرآیندهای DevOps دارد. با امکان ایجاد و حذف سریع ماشین‌های مجازی، مجازی‌سازی به تیم‌های توسعه این امکان را می‌دهد که به سرعت محیط‌های نرم‌افزاری مورد نیاز خود را راه‌اندازی و پس از اتمام کار، آن‌ها را به راحتی حذف کنند، که این امر منجر به صرفه‌جویی در هزینه‌ها و منابع می‌شود. علاوه بر این، مجازی‌سازی ریسک‌های مرتبط با استقرار نهایی در محیط تولید را کاهش داده و با ایجاد محیط‌های شبیه‌سازی شده برای آزمایش‌های پیش از استقرار، اطمینان حاصل می‌کند که نرم‌افزار قبل از راه‌اندازی به درستی کار می‌کند.

۲-۳-۲ کانتینرها

کانتینرها محیط‌هایی هستند که به برنامه‌های نرم‌افزاری امکان می‌دهند تا با تمام وابستگی‌های خود در یک بسته واحد جمع‌آوری شوند. آن‌ها همانند برنامه‌های نرم‌افزاری سنتی که به شما اجازه می‌دهند مستقل از نرم‌افزارهای دیگر و خود سیستم عامل کار کنند، نصب نمی‌شوند. مهم‌ترین دغدغه کانتینرها این است که چگونه محیطی فراهم کنند تا نرم‌افزارهایی که در یک محیط پردازشی اجرا می‌شوند با انتقال به محیط دیگر، بدون ایراد و مشکل اجرا شوند. این تکنولوژی از معماری میزبان بهره می‌برد تا از منابع سخت‌افزاری مشترک استفاده کند، اما اجرای برنامه‌ها را در یک محیط ایزوله و مستقل فراهم می‌کند. تمام اجزای ضروری مورد نیاز یک برنامه به صورت یک Image بسته‌بندی می‌شود. Image مربوطه در یک محیط ایزوله اجرا شده و فضای حافظه، CPU و فضای ذخیره‌سازی خود را با سیستم عامل به اشتراک نخواهد گذاشت. این عمل موجب می‌شود که فرآیندهای موجود در کانتینر، قادر به مشاهده سایر فرآیندها در خارج از آن نباشد.

کانتینرها و ماشین‌های مجازی هر دو ابزارهایی برای ایزوله‌سازی منابع نرم‌افزاری هستند، اما تفاوت‌های اساسی در معماری و کاربرد آن‌ها وجود دارد که در شکل ۲-۳ نشان داده شده است. ماشین‌های مجازی با ایجاد یک لایه انتزاعی کامل بر روی سخت‌افزار فیزیکی کار می‌کنند که به آن‌ها اجازه می‌دهد سیستم عامل‌های مستقل را بر روی هر VM اجرا کنند. این امر به هر ماشین مجازی امکان می‌دهد منابع سخت‌افزاری را به صورت مجزا استفاده کند، اما باعث می‌شود VM‌ها نسبت به کانتینرها سنگین‌تر و کم استفاده‌تر باشند. در مقابل، کانتینرها به جای سیستم عامل‌های کامل، تنها برنامه‌ها و وابستگی‌های خود را ایزوله می‌کنند و همگی بر روی هسته سیستم عامل میزبان اشتراکی اجرا می‌شوند، که این امر باعث سبک‌تر، سریع‌تر و



شکل ۲-۳: تفاوت ماشین‌های مجازی و کانتینر

مقیاس‌پذیرتر شدن کانتینرها نسبت به ماشین‌های مجازی می‌شود. از این‌رو، کانتینرها برای محیط‌هایی که نیازمند راه‌اندازی سریع و مدیریت منابع مانند میکروسرویس‌ها و برنامه‌های کاربردی مبتنی بر ابر هستند، ایده‌آل می‌باشند [۱۷]. در کنار مزایای فراوان کانتینرها، برخلاف ماشین‌های مجازی در امنیت و ایزو‌لاسیون داده‌ها محدودیت‌هایی دارند و ممکن است نیازمند ابزارهای پیچیده‌تر برای مدیریت لاغ‌ها و نظارت باشند، که می‌تواند پیاده‌سازی و نگهداری آن‌ها را چالش‌برانگیز سازد.

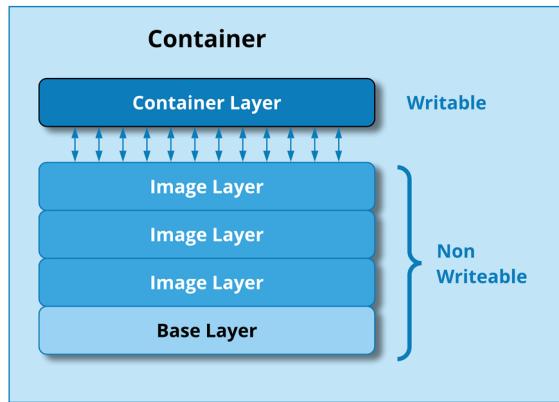
تکنولوژی کانتینر ریشه در مفهوم چارچوب‌های Unix مانند chroot دارد که در دهه ۱۹۷۰ معرفی شد. اما، پیشرفت‌های اصلی در این زمینه با ظهر Docker در سال ۲۰۱۳ آغاز شد. داکر یک پلتفرم متن‌باز است که استاندارد سازی ایجاد، اجرا و مدیریت کانتینرها را فراهم کرد و به سرعت به یکی از مهم‌ترین ابزارها در این حوزه تبدیل شد.

اجزای کلیدی مورد استفاده در پیاده‌سازی کانتینرها شامل موارد زیر است [۱۸]:

- موتورهای کانتینر^{۳۷}

- هماهنگ‌سازی کانتینر^{۳۸}

موتورهای کانتینری مانند Docker Engine و Containerd کانتینرها را ایجاد، اجرا و مدیریت می‌کنند. این موتورها از فناوری‌های موجود در هسته لینوکس مانند Namespaces و Control groups (cgroups) برای ایزو‌له‌سازی کانتینرها استفاده می‌کنند و به آن‌ها امکان می‌دهند که فرایندها و منابع سیستمی را به صورت مستقل از یکدیگر مدیریت کنند. بخشی از هسته لینوکس است که امکان جداسازی عناصری مثل شبکه، فرایندها و فضای فایل



شکل ۴-۲: معماری لایه‌ای تصویر داکر

سیستم را فراهم می‌کند. هر کانتینر در یک namespace جدگانه اجرا می‌شود که استقلال آن را نسبت به دیگر برنامه‌ها تضمین می‌کند. هر کانتینر در یک cgroups نیز به مدیریت استفاده از منابع سخت‌افزاری مانند CPU و حافظه توسط فرایندها کمک می‌کند. این فناوری امکان اختصاص دقیق منابع به کانتینرها را می‌دهد و از مصرف بیش از حد منابع توسط یک کانتینر جلوگیری می‌کند.

Docker Image به عنوان اساسی‌ترین بخش در اکوسیستم داکر نقش کلیدی در پیاده‌سازی و توزیع برنامه‌های نرم‌افزاری دارد. تصاویر داکر از یک معماری لایه‌ای بهره می‌برند. معماری لایه‌ای این امکان را فراهم می‌کند که تغییرات نسبت به یک تصویر پایه به صورت دیفرانسیلی اعمال شود. هر لایه در تصویر داکر، تغییراتی را نسبت به لایه قبلی اضافه می‌کند. این رویکرد باعث می‌شود که بازسازی و بهروزرسانی تصاویر کانتینری فقط بر روی لایه‌هایی که تغییر کرده‌اند انجام شود، که به نوبه خود باعث کاهش حجم داده‌های موردنیاز برای ذخیره‌سازی و انتقال می‌شود. زمانی که Dockerfile نوشته می‌شود، هر دستور (مانند COPY و RUN) یک لایه جدید در تصویر داکر ایجاد می‌کند. این لایه‌ها به ترتیبی که در داکرفایل آمده‌اند، روی هم اضافه می‌شوند. داکر از یک فایل سیستم Union استفاده می‌کند که به آن این اجازه را می‌دهد تا لایه‌های مختلف را به‌گونه‌ای ترکیب کند که به نظر یک فایل سیستم یکپارچه است [۱۹]. ساختار لایه‌ای تصویر داکر را در شکل ۴-۲ مشاهده می‌کنید.

برای مدیریت و مقیاس‌بندی کانتینرها در محیط‌های تولید، ابزارهای هماهنگ‌سازی مانند Docker و Kubernetes Swarm کاربرد دارند. این ابزارها به توسعه‌دهندگان این امکان را می‌دهند که خوشه^{۳۹}‌های بزرگ کانتینری را مدیریت کنند و برنامه‌ها را با انعطاف‌پذیری و دقت بالا مقیاس‌بندی نمایند.

۳-۳-۲ هماهنگ سازی کانتینرها (کوبرنتیز)

در دنیای توسعه نرم افزار، استفاده از معماری های مبتنی بر میکروسرویس ها و کانتینرها افزایش یافته است که هر دو نیازمند مدیریت دقیق و خودکار سرویس ها در محیط های تولید هستند. در محیط های پویا و با مقیاس بزرگ که دستگاه ها و خدمات به طور مکرر تغییر می کنند، تقریباً غیر ممکن است که با نیروی کار دستی، سرویسی با دسترسی بالا ارائه داد. در چنین شرایطی، ابزارهای هماهنگ سازی نقش حیاتی ایفا می کنند. آن ها به خودکارسازی مدیریت کانتینرها، مدیریت شبکه و نظارت بر سلامت سیستم کمک می کنند. بدون هماهنگ سازی تیم های توسعه و عملیات با چالش های عدیده ای از جمله کنترل ناموفق بر پیکربندی ها، مشکلات مربوط به برقراری ارتباط بین سرویس ها، دشواری های مربوط به مقیاس پذیری و برقراری تعادل بار مواجه می شوند. در میان ابزارهای هماهنگ سازی Kubernetes به عنوان یکی از پیشرفته بازار شناخته می شود که امکان مدیریت خودکار مجموعه های بزرگی از کانتینرها را فراهم می آورد. کوبرنتیز یک پلتفرم هماهنگ سازی کانتینر است که فرایند زمان بندی^{۴۰}، خودکارسازی استقرار، مدیریت و مقیاس گذاری اپلیکیشن های کانتینری را تسهیل می کند.

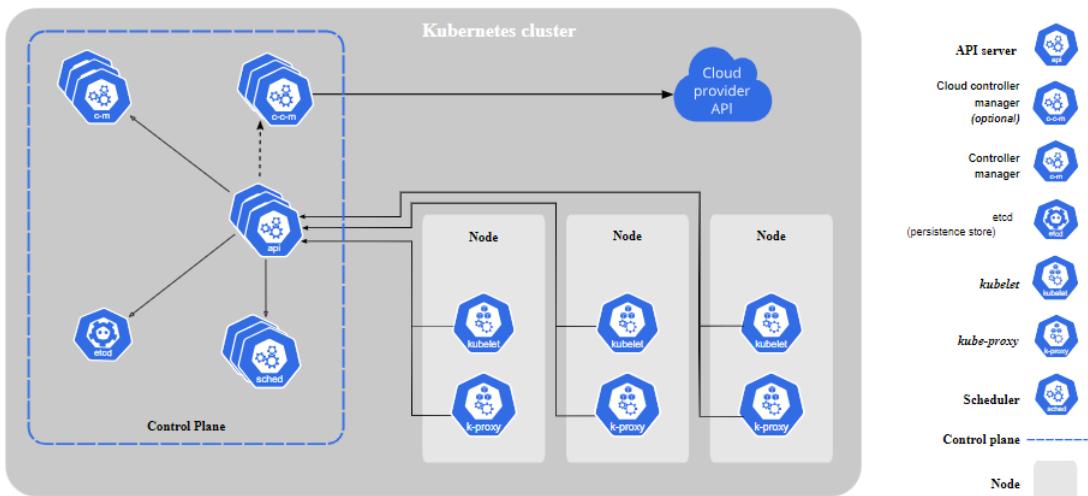
معماری کوبرنتیز

یک سیستم کوبرنتیز با تمام اجزای آن را یک خوش^{۴۱} می گویند. هر خوش شامل یک یا چند گره^{۴۲} است که می توانند فیزیکی^{۴۳} یا مجازی باشند. این گره ها به دو دسته اصلی یا همان سطح کنترل^{۴۴} و کاری^{۴۵} تقسیم می شوند. گره اصلی به عنوان مغز منطقی کوبرنتیز عمل می کند و وظایف مدیریتی خوش را بر عهده دارد. این گره شامل مولفه های اصلی زیر است:

- API Server: نقطه اصلی دریافت فرمان های کوبرنتیز به صورت REST^{۴۶} است و آن را پردازش می کند. این سرور مسئول اعتبارسنجی درخواست ها و اجرای آن ها بر روی خوش است. همچنین، این مولفه به عنوان بخشی از مولفه های دیگر گره اصلی عمل می کند تا اطمینان حاصل شود که دستورات به درستی اجرا می شوند.

- Scheduler: مولفه ای است که تصمیم می گیرد کدام پادها بر روی کدام گره های کاری قرار گیرند. این فرایند بر اساس منابع موجود و الزامات مشخص شده برای پادها صورت می گیرد. علاوه بر این، به طور مداوم وضعیت خوش را رصد می کند تا بهترین تصمیم ها را برای مکان یابی پادها بگیرد.

Scheduling ^{۴۰}	
Cluster ^{۴۱}	
Node ^{۴۲}	
Bare-metal ^{۴۳}	
Control plane ^{۴۴}	
Worker ^{۴۵}	
Representational State Transfer ^{۴۶}	



شکل ۵-۲: مولفه‌های یک خوشه کوبرنتیز

• **Controller Managers**: مجموعه‌ای از فرآیندهایی است که حلقه‌های نظارتی را اجرا می‌کنند. این کنترل‌کننده‌ها

وضعیت خوشه را با حالت مطلوب مطابقت می‌دهند. به عنوان مثال، اگر یک پاد از کار افتاده باشد، یک کنترل‌کننده

وظیفه دارد تا یک پاد جدید را برای جایگزینی ایجاد کند.

• **etcd**: یک پایگاه داده توزیع شده است که تمام داده‌های مهم از جمله وضعیت خوشه در هر لحظه، پیکربندی خوشه،

اطلاعات مربوط به هر گره و کانتینترهای درون آن را در خود ذخیره می‌کند.

گره‌های کاری نیز پادهای اپلیکیشن‌های کاربر را بر عهده دارند. این گره‌ها شامل مولفه‌های زیر هستند:

• **Kubelet**: این مولفه وظیفه مدیریت سلامت پادها را بر عهده دارد. علاوه بر این اطمینان حاصل می‌کند که کانتینترها در

پادها بر اساس تنظیمات مشخص شده اجرا شوند و با API Server ارتباط برقرار کند تا وضعیت را به روز رسانی کند.

• **Kube-proxy**: وظیفه مدیریت ترافیک شبکه درون خوشه را بر عهده دارد. این مولفه ارتباطات شبکه بین کانتینترها را

تسهیل می‌کند و از قوانین IPTables برای مسیریابی ترافیک استفاده می‌کند.

پاد و Deployment

پاد در کوبرنتیز کوچک‌ترین واحد قابل استقرار و مدیریت است که شامل یک یا چند کانتینتر می‌شود. این کانتینترها به طور

معمول از یک فضای ذخیره‌سازی و شبکه مشترک استفاده می‌کنند و می‌توانند به راحتی با یکدیگر ارتباط برقرار کنند. هر پاد

دارای یک آدرس IP یکتا است که امکان ارتباط مستقیم بین کانتینترها را فراهم می‌کند. پادها به صورت موقت طراحی شده‌اند و

ممکن است به دلایل مختلفی مانند بروزرسانی یا مشکلات سیستمی حذف و بازسازی شوند. معمولاً پادها برای اجرای یک

برنامه خاص (مانند یک سرور وب) استفاده می‌شوند، اما در مواردی که نیاز به همکاری نزدیک کاتینترها باشد، می‌توانند چندین کاتینتر مرتبط را نیز شامل شوند.

در کوبرنتیز ابزاری قدرتمند برای مدیریت و مقیاس‌پذیری برنامه‌ها است. این ابزار به کاربران امکان می‌دهد تا مجموعه‌ای از پادها را تعریف و به سادگی مدیریت کنند. کاربران می‌توانند نسخه‌های جدید برنامه‌ها را منتشر کرده، مقیاس آن‌ها را تنظیم و در صورت بروز مشکل به نسخه‌های قبلی بازگردند. همچنین، به طور خودکار وضعیت پادها را مانیتور می‌کند و در صورت نیاز، آن‌ها را مجدداً راه‌اندازی می‌کند تا اطمینان حاصل شود که همیشه تعداد مشخصی از پادها در حال اجرا هستند. این ویژگی‌ها آن را به گزینه‌ای مناسب برای مدیریت برنامه‌های پایدار و قابل اعتماد در محیط‌های تولیدی تبدیل می‌کند.

مدیریت داده

در کوبرنتیز، Volume مکانیزمی است که به پادها اجازه می‌دهد به داده‌های پایدار دسترسی داشته باشند. هر Volume به یک پاد متصل شده حتی با پایان یافتن پاد، داده‌ها می‌توانند حفظ شوند. انواع مختلفی از Volume‌ها وجود دارد که عبارتند از:

- emptyDir: این نوع Volume رمانی که پاد ایجاد می‌شود، ساخته شده و هنگام حذف پاد، از بین می‌رود. برای ذخیره‌سازی موقتی مناسب است.

- hostPath: به فایل یا دایرکتوری روی گره میزبان اشاره می‌کند. این گزینه به مواردی که به دسترسی مستقیم به منابع میزبان نیاز دارند، کمک می‌کند.

- PersistentVolume (PV), PersistentVolumeClaim (PVC): از اجزای اصلی برای مدیریت داده‌های پایدار هستند. PV به عنوان منبع ذخیره‌سازی مستقل ایجاد می‌شود و شامل ویژگی‌هایی مانند ظرفیت و حالت‌های دسترسی است. PVC یک درخواست از طرف کاربر برای استفاده از PV است. PVC‌ها به طور خودکار به PV‌های موجود متصل می‌شوند که شرایط مورد نیاز را برآورده کنند.

- configMap: برای وارد کردن داده‌های پیکربندی به عنوان فایل‌ها در پادها استفاده می‌شود.
- secret: برای ذخیره داده‌های حساس مانند رمزهای عبور به صورت رمزگذاری شده و دسترسی به آن‌ها به عنوان فایل در پادها.

فصل ۳

MLOps بر مفاهیم

۱-۳ مقدمه

در حالی که مدل‌های یادگیری ماشین به طور گستردگی توسعه یافته‌اند، انتقال آن‌ها از مفهوم آزمایشی به محیط تولید اغلب با شکست مواجه می‌شود. این فاصله بیشتر به خاطر این است که تاکنون توجه اصلی روی ساخت مدل‌ها بوده است، نه روی تولید محصولات یادگیری ماشین که قابلیت استفاده در محیط تولید را دارند. علاوه بر آن، مدیریت بخش‌ها و زیرساخت‌های پیچیده‌ای که برای یک استقرار مؤثر ضروری هستند نیز در این امر مغفول مانده‌اند. برای رفع این مسئله، مفهوم عملیات یادگیری ماشین یا MLOps معرفی شده است. MLOps بر روی خودکارسازی و عملیاتی کردن فرآیندهای یادگیری ماشین تمرکز دارد تا انتقال پروژه‌های یادگیری ماشین از مفهوم به تولید را تسهیل کند. این رویکرد شامل دیدگاه جامعی از طراحی سیستم، هماهنگی اجزا، تعریف نقش‌ها و مسئولیت‌ها می‌باشد. هدف کاهش خطأ به‌منظور افزایش قابلیت اطمینان و کارایی سیستم‌های یادگیری ماشین در کاربردهای واقعی می‌باشد. این فصل به بررسی تعریف، اصول، ابزار و معماری جامعی از یک پلتفرم MLOps پرداخته و در نهایت، محصولات و رقبا این حوزه را بررسی می‌کند.

۲-۳ تعریف مفاهیم اولیه

MLOps یا پیاده‌سازی یادگیری ماشین به مجموعه‌ای از فرآیندها، ابزارها و شیوه‌ها جهت مدیریت چرخه توسعه مدل‌های یادگیری ماشین در یک محیط عملیاتی اشاره دارد. همچنین این چرخه شامل همکاری بین دانشمندان داده و مهندسان است به‌گونه‌ای که این اطمینان حاصل شود که مدل‌ها به‌طور مؤثر توسعه، استقرار، پایش و بهروزرسانی می‌شوند. DevOps هدف MLOps افزایش سرعت، قابلیت اعتماد و مقیاس‌پذیری مدل‌های یادگیری ماشین و فرآیند توسعه این مدل‌ها در تولید

است؛ در حالی که خطرات ناشی از ریسک عدم موفقیت را نیز کاهش می‌دهد. همچنین به کارگیری MLOps فرآیند مدیریت را ساده‌تر کرده، کیفیت را افزایش می‌دهد و استقرار مدل‌های یادگیری عمیق و یادگیری ماشین در محیط‌های تولید با مقیاس بزرگ را خودکار می‌کند. لذا می‌توان گفت یکی از اهداف MLOps، بهبود خودکارسازی و ارتقای کیفیت مدل‌های تولید و در عین حال توجه به الزامات تجاری و نظارتی است.

استقرار مدل‌های یادگیری ماشین روی محیط عملیاتی در MLOps اهمیت زیادی دارد، زیرا به سازمان‌ها کمک می‌کند تا مطمئن شوند که مدل‌هایی‌شان در طول زمان دقیق، قابل اعتماد و کارآمد هستند. به طورکلی، MLOps با خودکار کردن بسیاری از مراحل مربوط به استقرار و مدیریت مدل‌های یادگیری ماشین، به دانشمندان و مهندسان داده اجازه می‌دهد تا با همکاری یکدیگر به ارائه سریع‌تر و کارآمدتر مدل‌های یادگیری ماشین دست یابند.

۱-۲-۳ اصول

برای تسهیل در رسیدن به اهداف فوق، تیم‌های MLOps از اصول زیر استفاده می‌کنند:

۱. خط لوله خودکار CI/CD و هماهنگ‌سازی جریان کار^۱: خودکارسازی CI/CD شامل مراحل ساخت، آزمایش، تحویل و استقرار است که به توسعه‌دهندگان نسبت به موفقیت یا شکست مراحل مختلف بازخورد سریعی را ارائه داده و بهره‌وری کلی را افزایش می‌دهد [۲۰]. در همین حال، هماهنگ‌سازی جریان کاری وظایف یک خط لوله یادگیری ماشین را با استفاده از گراف‌های بدون حلقه‌ی جهت دار^۲ هماهنگ می‌کند، که ترتیب اجرای وظایف را با توجه به روابط و وابستگی‌ها تعیین می‌کند. ترکیب این دو رویکرد می‌تواند به بهبود عملکرد و کارایی تیم‌های توسعه و داده‌کاوی کمک کند [۲۱، ۲۲].

۲. کنترل نسخه مدل‌های یادگیری ماشین، مجموعه‌داده‌ها و کد منبع: با استفاده از نسخه‌بندی مدل، داده و کد منبع، می‌توان هر تغییر و اصلاحی را در طول زمان دنبال کرد، که این امر به توسعه‌دهندگان و محققان اجازه می‌دهد تا به راحتی به نسخه‌های قبلی بازگردند و نتایج را بازبینی کنند. این قابلیت برای حفظ یکپارچگی و شفافیت در پروژه‌های نرم‌افزاری و علمی بسیار حیاتی است [۲۰].

۳. نظارت و آموزش مداوم مدل یادگیری ماشین: آموزش مداوم^۳ در یادگیری ماشین به معنای آموزش دوره‌ای مدل‌های یادگیری ماشین بر اساس داده‌های جدید است. این فرآیند همیشه شامل یک مرحله ارزیابی برای سنجش تغییرات

Workflow^۱
Directed Acyclic Graph (DAG)^۲
Continuous Training (CT)^۳

کیفیت مدل است [۲۳]. نظارت مداوم به معنای ارزیابی دوره‌ای داده‌ها، مدل‌ها (مانند دقت پیش‌بینی)، کد منبع و متابع زیرساختی است تا خطاهای یا تغییرات احتمالی که بر کیفیت محصول تأثیر می‌گذارند، شناسایی شوند. این فرآیند به توسعه‌دهنگان امکان می‌دهد تا به سرعت مشکلات را شناسایی و بر طرف کنند و از افت عملکرد مدل جلوگیری کنند. یکی از دلایل لزوم آموزش مداوم، رانش داده یا مدل^۴ است، که به تغییرات تدریجی در داده‌ها یا عملکرد مدل در طول زمان اشاره دارد و می‌تواند باعث کاهش دقت پیش‌بینی‌ها شود [۲۴]. این اصل در MLOps برای اطمینان از عملکرد بهینه مدل‌ها و واکنش سریع به تغییرات محیطی و داده‌ها ضروری است. این فرآیند بهره‌وری را افزایش می‌دهد و کیفیت کلی سیستم‌های یادگیری ماشین را بهبود می‌بخشد. در نهایت، ترکیب آموزش و نظارت مداوم به توسعه‌دهنگان کمک می‌کند تا مدل‌ها را به روز نگه داشته و از تأثیرات منفی رانش داده یا مدل جلوگیری کنند [۲۵].

۴. ثبت فراداده^۵ یادگیری ماشین: ثبت فراداده برای هر مرحله در جریان کار یادگیری ماشین شامل ثبت جزئیات هر دوره آموزش مدل، مانند تاریخ و زمان آموزش، مدت زمان، پارامترهای استفاده شده و معیارهای عملکرد مدل می‌باشد [۲۶]. علاوه بر این، جزئیات مدل که شامل داده‌ها و کدهای استفاده شده است، باید ثبت شود تا قابلیت پیگیری کامل آزمایشات فراهم گردد. این امر به توسعه‌دهنگان کمک می‌کند تا تغییرات و نتایج را به دقت مستند کرده و در صورت نیاز به نسخه‌های قبلی بازگردند [۲۶].

۵. حلقه‌های بازخورد^۶: حلقه‌های بازخورد به توسعه‌دهنگان اجازه می‌دهند تا به طور مداوم مدل‌ها را بهبود بخشنند، مشکلات را شناسایی و رفع کنند و از افت کیفیت جلوگیری کنند. این رویکرد به تضمین کیفیت و کارایی مدل‌های یادگیری ماشین کمک می‌کند و فرآیند توسعه را به یک چرخه تکراری و قابل بهبود تبدیل می‌کند که به سرعت به تغییرات و نیازهای جدید پاسخ می‌دهد [۲۶]. به عنوان مثال، یک حلقه بازخورد از مرحله مهندسی مدل آزمایشی به مرحله قبلی مهندسی ویژگی می‌تواند بسیار مفید باشد.

می‌توان اضافه کرد که یکی از اصول مهم که کمتر جنبه فنی دارد و در روح فرهنگی DevOps نیز جایگاه ویژه‌ای دارد، اصل همکاری^۷ است. این اصل بر امکان همکاری مسترک افراد بر روی داده‌ها، مدل‌ها و کدها تأکید دارد. علاوه بر جنبه‌های فنی، اصل همکاری به ایجاد فرهنگ کاری مشارکتی توجه دارد که هدف آن کاهش ایزوله‌سازی‌های حوزه‌ای بین نقش‌های مختلف است. چنین رویکردی باعث می‌شود تا افراد با تخصص‌های گوناگون به طور هم‌افزا با یکدیگر کار کنند، دانش خود را به اشتراک بگذارند و از هم بیاموزند.

Data or Model Drift^۴Metadata^۵feedback loops^۶Collaboration^۷

۲-۲-۳ اجزاء

پس از شناسایی اصولی که در قسمت قبل صحبت کردیم، اکنون اجزای دقیق یک معماری MLOps را توضیح داده و ارتباط هرکدام با اصول گفته شده را بیان می‌کنیم.

هماهنگ‌سازی جریان‌کاری

هماهنگ‌سازی جریان‌کاری^۸ به عنوان یکی از اجزای حیاتی در مدیریت و خودکارسازی جریان‌های کاری پیچیده در حوزه‌های مختلف از جمله یادگیری ماشین و مهندسی داده، نقش مهمی ایفا می‌کند. این سیستم‌ها مانند شکل ۱-۳ از گراف‌های بدون حلقه جهت‌دار برای نمایش ترتیب اجرای وظایف استفاده می‌کنند. هر مرحله از این جریان‌کاری ممکن است شامل استخراج داده، آموزش مدل یا استنتاج باشد. این سیستم‌ها نه تنها ترتیب اجرای وظایف را مدیریت می‌کنند، بلکه وابستگی‌های متقابل بین وظایف را نیز مورد توجه قرار می‌دهند. همچنین این ابزارها به کاربران امکان می‌دهند تا جریان‌های کاری را به صورت خودکار و مقیاس‌پذیر اجرا کنند. این امر به ویژه در محیط‌های بزرگ با داده‌های کلان اهمیت دارد [۲۲].

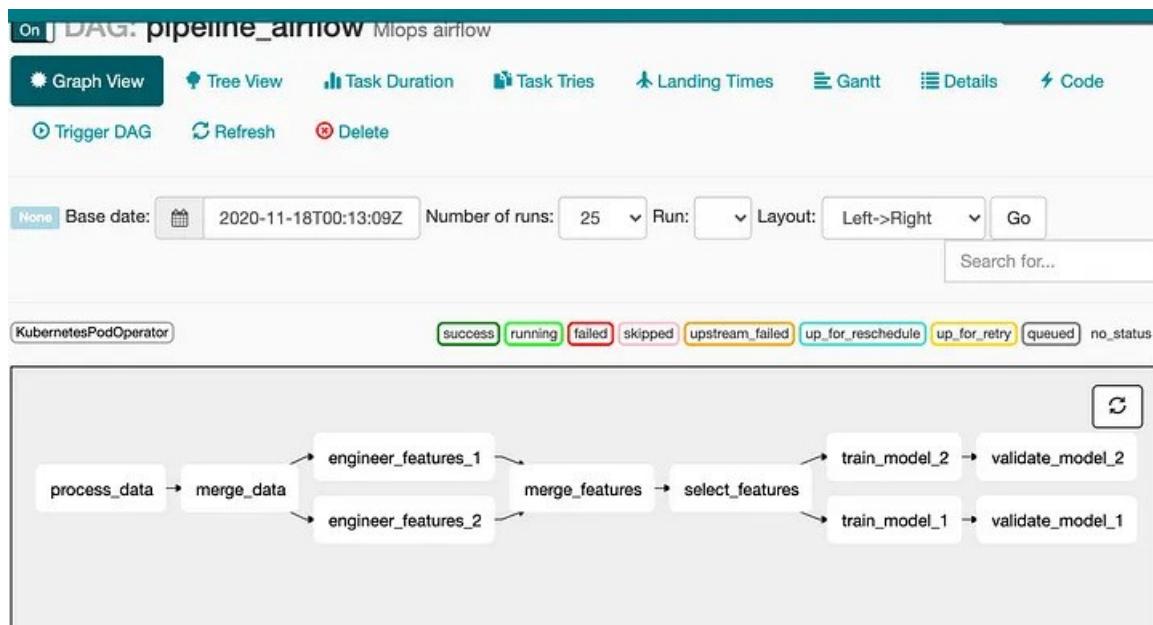
ابزارهای متن‌باز معروف در زمینه یادگیری ماشین Apache Airflow pipeline [۲۷] و Kubeflow [۲۸] نیز بهتر برای استخراج، تبدیل و بارگذاری^۹ داده‌های بزرگ استفاده می‌کنند. Apache Airflow Pipelines بخشی از پلتفرم Kubeflow [۲۹] است که برای اجرای جریان‌های کاری یادگیری ماشین بر روی کوبرنتیز طراحی شده است. از این ابزار به طور خاص برای توسعه و استقرار مدل‌های یادگیری ماشین در محیط‌های ابری مناسب است که در فصل‌های بعدی با آن بیشتر آشنا خواهیم شد.

انبار ویژگی

انبار ویژگی^{۱۰} یک سیستم مدیریت داده است که به منظور ذخیره‌سازی، مدیریت و اشتراک‌گذاری ویژگی‌های مورد استفاده در مدل‌های یادگیری ماشین طراحی شده است. این سیستم (شکل ۲-۳) دارای دو بخش اصلی است: پایگاه داده آنلاین و پایگاه داده آفلاین. هر یک از این پایگاه‌های داده نقش خاصی در فرآیند مدیریت و استفاده از ویژگی‌ها ایفا می‌کنند.

پایگاه داده آفلاین برای ذخیره‌سازی و مدیریت ویژگی‌هایی استفاده می‌شود که در فرآیندهای آزمایش و تحلیل به کار می‌روند. این پایگاه داده معمولاً با تأخیر نسبتاً بیشتری نسبت به پایگاه داده آنلاین استفاده می‌شود و برای مواردی مناسب است

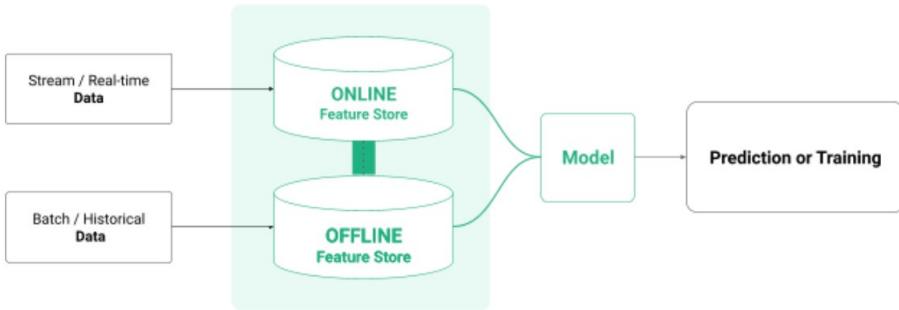
Workflow Orchestration^۸
Extract, Transform, Load (ETL)^۹
Feature Store^{۱۰}



شکل ۳: خط لوله در Apache Airflow

که نیاز به پردازش حجم زیادی از داده‌ها در مدت زمان طولانی‌تر دارند. ویژگی‌هایی که در این پایگاه داده ذخیره می‌شوند، اغلب در فرآیندهای آموزش مدل‌های یادگیری ماشین مورد استفاده قرار می‌گیرند.

پایگاه داده آنلاین برای ارائه ویژگی‌ها به صورت بلاذرنگ استفاده می‌شود و تأخیر کمی دارد. این پایگاه داده‌ها برای سیستم‌هایی مناسب هستند که نیاز به پاسخگویی سریع دارند. زمانی که یک مدل یادگیری ماشین نیاز به استفاده از ویژگی‌ها برای انجام پیش‌بینی‌های فوری دارد، داده‌ها از این پایگاه داده آنلاین بازیابی می‌شوند. این نوع پایگاه داده‌ها باید توانایی پشتیبانی از حجم بالای درخواست‌ها را داشته باشند تا بتوانند عملکرد مطلوبی را در شرایط عملیاتی فراهم کنند. ویژگی‌هایی که در این پایگاه داده ذخیره می‌شوند، اغلب در فرآیندهای استنتاج مدل‌های یادگیری ماشین مورد استفاده قرار می‌گیرند. با استفاده از انباره ویژگی توسعه‌دهندگان می‌توانند ویژگی‌های از پیش پردازش شده را به صورت مرکز ذخیره کرده و به راحتی در پروژه‌های مختلف به اشتراک بگذارند، که این امر به تسريع فرآیند توسعه مدل‌ها و بهبود دقت پیش‌بینی‌ها کمک می‌کند. این سیستم‌ها معمولاً بر روی زیرساخت‌های ابری اجرا می‌شوند تا مقیاس‌پذیری بالا و کارایی مورد نیاز برای پردازش داده‌های کلان را فراهم کنند [۲۵]. از ابزار معروف متن‌باز می‌توان به Feast [۳۰] اشاره نمود.



شکل ۲-۳: انباره ویژگی

بانک مدل

بانک مدل^{۱۱} یکی از ابزارهای بسیار مهم در مدیریت مدل‌های یادگیری ماشین است که به تیم‌ها کمک می‌کند تا مدل‌های خود را به صورت سازماندهی شده ذخیره، مدیریت و ردیابی کنند. همچنین اطلاعات مربوط به هر مدل را از جمله نسخه، تاریخ آخرین آموزش، معیارهای ارزیابی و مستندات مربوطه را نگهداری می‌کند. این امر به تیم‌ها کمک می‌کند تا با استفاده از نسخه‌های مختلف مدل‌ها، آزمایش‌های مختلفی انجام دهند و بهترین مدل را انتخاب کنند. همچنین به هنگام بروز مشکل در مدل‌های جدید، می‌توان از مدل‌های قابل قبول قبلی برای محیط عملیاتی استفاده کرد [۳۱]. از ابزارهای معروف متن‌باز می‌توان به [۳۲]MLflow اشاره کرد.

انبار فراداده

انبار فراداده یادگیری ماشین^{۱۲} برای پیگیری و ذخیره‌سازی اطلاعات مربوط به هر مرحله از جریان کاری یادگیری ماشین استفاده می‌شود. فراداده‌ها می‌توانند شامل جزئیاتی نظیر تاریخ و زمان آموزش مدل، مدت زمان هر مرحله از آموزش، پارامترهای استفاده شده، معیارهای عملکرد مدل، و سلسله‌مراتب مدل (مثل داده‌ها و کدهای استفاده شده) باشند. یکی از کاربردهای اصلی انبار فراداده‌ها، مدیریت کارآمد پروژه‌های پیچیده یادگیری ماشین است [۲۶]. به عنوان مثال، در پروژه‌های بزرگ که شامل آزمایش‌ها و مدل‌های متعددی هستند، پیگیری دقیق و منظم فراداده‌ها می‌تواند به تیم‌ها کمک کند تا نتایج قبلی را به راحتی بازبینی کنند، مشکلات را شناسایی کنند و بهینه‌سازی‌های لازم را انجام دهند. MLflow یک ابزار معروف برای یک سیستم پیشرفته مدیریت فراداده است که همراه با بانک مدل امکان مدیریت یکپارچه مدل‌ها و فراداده‌ها را فراهم می‌کند.

Model Registry^{۱۱}
ML Metadata Store^{۱۲}

استقرار مدل

استقرار مدل^{۱۳} به فرآیندی اشاره دارد که در آن مدل‌های یادگیری ماشین آماده برای استفاده، به کار گرفته می‌شوند تا به صورت عملیاتی به پیش‌بینی‌ها و استنتاج‌ها پردازند. این فرآیند برای تبدیل مدل‌های آموزشی به ابزارهای قابل استفاده در محیط‌های تولیدی ضروری است و می‌تواند به صورت آنلاین برای پیش‌بینی‌های بلادرنگ یا به صورت دسته‌ای^{۱۴} برای پردازش حجم بالای داده‌ها پیاده‌سازی شود. در محیط‌های عملیاتی، فرآیند استقرار مدل به سه شکل اصلی بلادرنگ، دسته‌ای و بدون سرور پیاده‌سازی می‌شود [۳۱].

در استنتاج بلادرنگ^{۱۵}، مدل‌های یادگیری ماشین به گونه‌ای پیاده‌سازی می‌شوند که بتوانند به سرعت و با کمترین تأخیر ممکن پیش‌بینی‌ها را انجام دهند. این نوع استنتاج برای کاربردهایی نظیر سیستم‌های توصیه‌گر، تحلیل داده‌های حسگرهای برنامه‌های کاربردی که نیاز به پاسخ‌های سریع دارند، مناسب است. به عنوان مثال، در سیستم‌های پیشنهاددهی محتوا مانند نتفلیکس یا آمازون، مدل‌ها باید به صورت بلادرنگ تحلیل کنند و پیشنهادهای شخصی‌سازی شده را ارائه دهند. تکنولوژی‌هایی مانند RESTful APIs و gRPC معمولاً برای پیاده‌سازی این نوع سرویس‌دهی استفاده می‌شوند.

استنتاج دسته‌ای^{۱۶} برای پردازش حجم وسیعی از داده‌ها به کار می‌رود که معمولاً به صورت زمان‌بندی شده انجام می‌شود. این روش برای تحلیل داده‌های کلان و پردازش‌های بزرگ مناسب است. به عنوان مثال، در تجزیه و تحلیل رفتار مشتریان یک فروشگاه آنلاین، داده‌های خریدهای گذشته می‌تواند به صورت دسته‌ای پردازش شود تا الگوهای مختلف شناسایی شود. ابزارهایی مانند Apache Spark [۳۳] و Hadoop MapReduce [۳۴] معمولاً برای پیاده‌سازی استنتاج دسته‌ای استفاده می‌شوند.

در استنتاج بدون سرور^{۱۷}، مدل‌ها به صورت پویا و بر اساس تقاضا اجرا می‌شوند که هزینه و مقیاس‌پذیری را بهینه می‌کند. این نوع استنتاج زمانی مورد استفاده قرار می‌گیرد که نیاز به سرویس‌دهی مقیاس‌پذیر و مقرن به صرفه باشد. در استنتاج بدون سرور، مدل‌ها فقط زمانی که لازم است اجرا می‌شوند و بنابراین منابع بهینه‌سازی می‌شوند. سرویس‌های ابری مانند AWS Lambda و Google Cloud Functions معمولاً برای پیاده‌سازی این نوع استنتاج استفاده می‌شوند. از ابزارهای معروف متن‌باز برای استقرار مدل می‌توان به Knative [۳۵] اشاره کرد.

Model Serving ^{۱۳}	
Batch ^{۱۴}	
Real-time Inference ^{۱۵}	
Batch Inference ^{۱۶}	
Serverless Inference ^{۱۷}	

ناظارت

ناظارت^{۱۸} در یادگیری ماشین یکی از مؤلفه‌های حیاتی برای تضمین عملکرد بهینه مدل‌ها و زیرساخت‌های مرتبط است. ناظارت مداوم بر مدل‌های یادگیری ماشین به دلایلی از جمله اطمینان از دقت پیش‌بینی‌ها، شناسایی ناهنجاری‌ها و بهبود مداوم عملکرد مدل‌ها ضروری است [۳۶]. ابزارهایی مانند TensorBoard و MLflow نیز نقش مهمی در ناظارت بر مدل‌های یادگیری ماشین ایفا می‌کنند. TensorBoard بتویژه برای مصورسازی و تحلیل مراحل مختلف آموزش مدل‌ها مفید است.

ناظارت در یادگیری ماشین تنها به مدل‌ها محدود نمی‌شود؛ بلکه زیرساخت‌های مرتبط با یادگیری ماشین نیز نیاز به ناظارت دارند. این ناظارت شامل ناظارت بر فرآیندهای CI/CD، هماهنگی سرویس‌ها، خوش‌های عملیاتی کوبرنتیز و گره‌های محاسباتی می‌شود [۲۶]. یکی از ابزارهای رایج برای ناظارت، Prometheus است که به همراه Grafana برای مصورسازی داده‌ها استفاده می‌شود. علاوه بر این، پشته ELK (Kibana، Logstash، Elasticsearch) نیز یک مجموعه قدرتمند برای جستجو، تحلیل و مصورسازی لاگ‌های سیستم است که می‌تواند به شناسایی و رفع سریع مشکلات کمک کند. ابزارهای ناظارتی به مهندسان اجازه می‌دهند تا هرگونه ناهنجاری در زیرساخت‌ها را به سرعت شناسایی و رفع کنند، که این امر موجب کاهش زمان از کار افتادگی سیستم و افزایش بهره‌وری می‌شود.

زیرساخت آموزش و استقرار مدل

این زیرساخت شامل منابع محاسباتی اصلی مانند واحد پردازش مرکزی، حافظه و واحد پردازش گرافیکی^{۱۹} است که برای پردازش داده‌ها و اجرای الگوریتم‌های پیچیده می‌باشد. زیرساخت‌ها می‌توانند به دو شکل توزیع شده^{۲۰} و غیرتوزیع شده پیاده‌سازی شوند. زیرساخت‌های غیرتوزیع شده معمولاً شامل ماشین‌های محلی هستند که با وجود سادگی در پیاده‌سازی، محدودیت‌هایی در مقیاس‌پذیری دارند. از سوی دیگر، زیرساخت‌های توزیع شده که معمولاً در بستر محاسبات ابری اجرا می‌شوند، امکان توزیع بار کاری بین چندین گره محاسباتی را فراهم می‌کنند و از این طریق مقیاس‌پذیری و کارایی بالاتری ارائه می‌دهند. یکی از ابزارهای محبوب برای مدیریت و هماهنگ‌سازی محاسبات توزیع شده، کوبرنتیز است که امکان مدیریت کاتینرها و توزیع بار کاری بین گره‌ها را فراهم می‌کند. همچنین Red Hat OpenShift نیز به عنوان یک پلتفرم دیگر شناخته می‌شود که قابلیت‌های مشابهی ارائه می‌دهد [۳۱].

برای بهینه‌سازی عملکرد مدل‌های یادگیری عمیق، استفاده از واحد پردازش گرافیکی که برای ضرب ماتریسی

Monitoring^{۱۸}
GPU^{۱۹}
Distributed^{۲۰}

بهینه‌سازی شده‌اند، استفاده می‌شود. در دستگاه‌های لبه^{۲۱} به دلیل محدودیت‌های فضای توان محاسباتی و مصرف انرژی، اجرای مدل‌های یادگیری عمیق پیچیده با چالش‌های خاصی مواجه است. برای غلبه بر این محدودیت‌ها و بهینه‌سازی عملکرد مدل‌ها در این دستگاه‌ها، تکنیک‌های مختلفی مورد استفاده قرار می‌گیرد. یکی از این تکنیک‌ها، استفاده از شبکه‌های عصبی کوانتیزه شده است. در کوانتیزاسیون، وزن‌ها و محاسبات شبکه عصبی از دقت کامل (به عنوان مثال، اعداد با دقت ۳۲ بیت) به اعداد با دقت پایین‌تر (مانند ۸ بیت یا حتی کمتر) کاهش می‌یابند. این کاهش دقت باعث کاهش حجم مدل و کاهش نیاز به منابع محاسباتی می‌شود. علاوه بر این، با استفاده از عملیات نقطه شناور کم‌دقت، می‌توان محاسبات را سریع‌تر و با مصرف انرژی کمتری انجام داد. تکنیک دیگر، هرس کردن^{۲۲} است که شامل حذف اتصالات غیرضروری و وزن‌های کوچک در شبکه عصبی می‌شود. این فرآیند باعث کاهش تعداد پارامترهای مدل می‌شود، بدون آنکه تأثیر قابل توجهی بر دقت مدل بگذارد. هرس کردن مدل را سبک‌تر و اجرای آن را سریع‌تر می‌کند، که این امر برای دستگاه‌های لبه با منابع محدود بسیار مفید است.

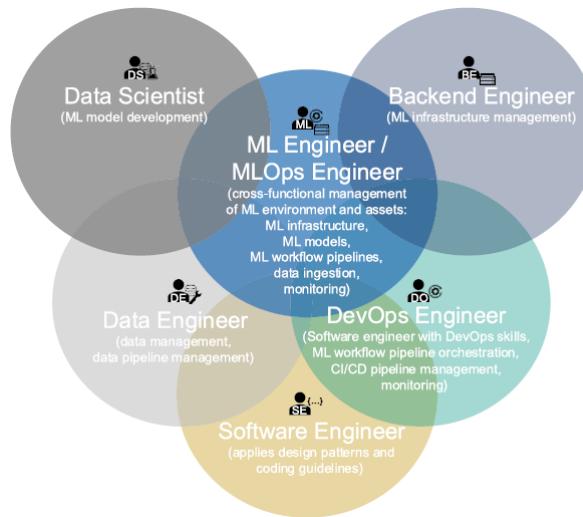
مخزن کد منبع

مخزن کد منبع به عنوان یک نقطه مشترک برای نگهداری و مدیریت کدهای مربوط به مدل‌های یادگیری ماشین یک سازمان عمل می‌کند. با استفاده از سیستم‌های مدیریت نسخه مانند گیت، تیم‌ها می‌توانند به راحتی تغییرات کد را پیگیری کرده و در صورت لزوم به نسخه‌های قبلی کد بازگردند. این مخزن همچنین به خودکارسازی فرآیند CI/CD کمک می‌کند، به طوری که هرگونه تغییر در کد به طور خودکار خط لوله را فعال کرده و تغییرات تست، ارزیابی و در محیط‌های مختلف مستقر می‌شوند. از ابزارهای متن‌باز برای پیاده‌سازی آن می‌توان به GitLab^[۳۷] و Gerrit^[۳۸] اشاره نمود.

خط لوله CI/CD

همان‌طور که در گذشته نیز راجع به آن صحبت کردیم، خط لوله CI/CD به تیم‌ها اجازه می‌دهد تا کدهای مدل و داده‌ها را به صورت مداوم تست، تأیید و استقرار دهند. در این فرآیند، مدل‌ها به طور خودکار بازآموزی و بهبود می‌یابند و در محیط‌های مختلف (توسعه، تست، تولید) به صورت پیوسته به روزرسانی می‌شوند. این کار نه تنها باعث افزایش کیفیت و دقت مدل‌ها می‌شود بلکه زمان توسعه و عرضه را نیز به طرز قابل توجهی کاهش می‌دهد. در MLOps این خط لوله‌ها در مراحل مختلف از جمله آموزش مدل، ارزیابی، استقرار و نظارت بر عملکرد مدل‌ها و همچنین داده‌ها استفاده می‌شوند [۲۶]. از ابزارهای مناسب برای این کار می‌توان به Jenkins^[۳۹] و GitLab CI^[۳۷] اشاره کرد.

^{۲۱} Edge Devices
^{۲۲} Pruning



شکل ۳-۳: نقش‌ها و اشتراکات آن‌ها در پارادایم MLOps

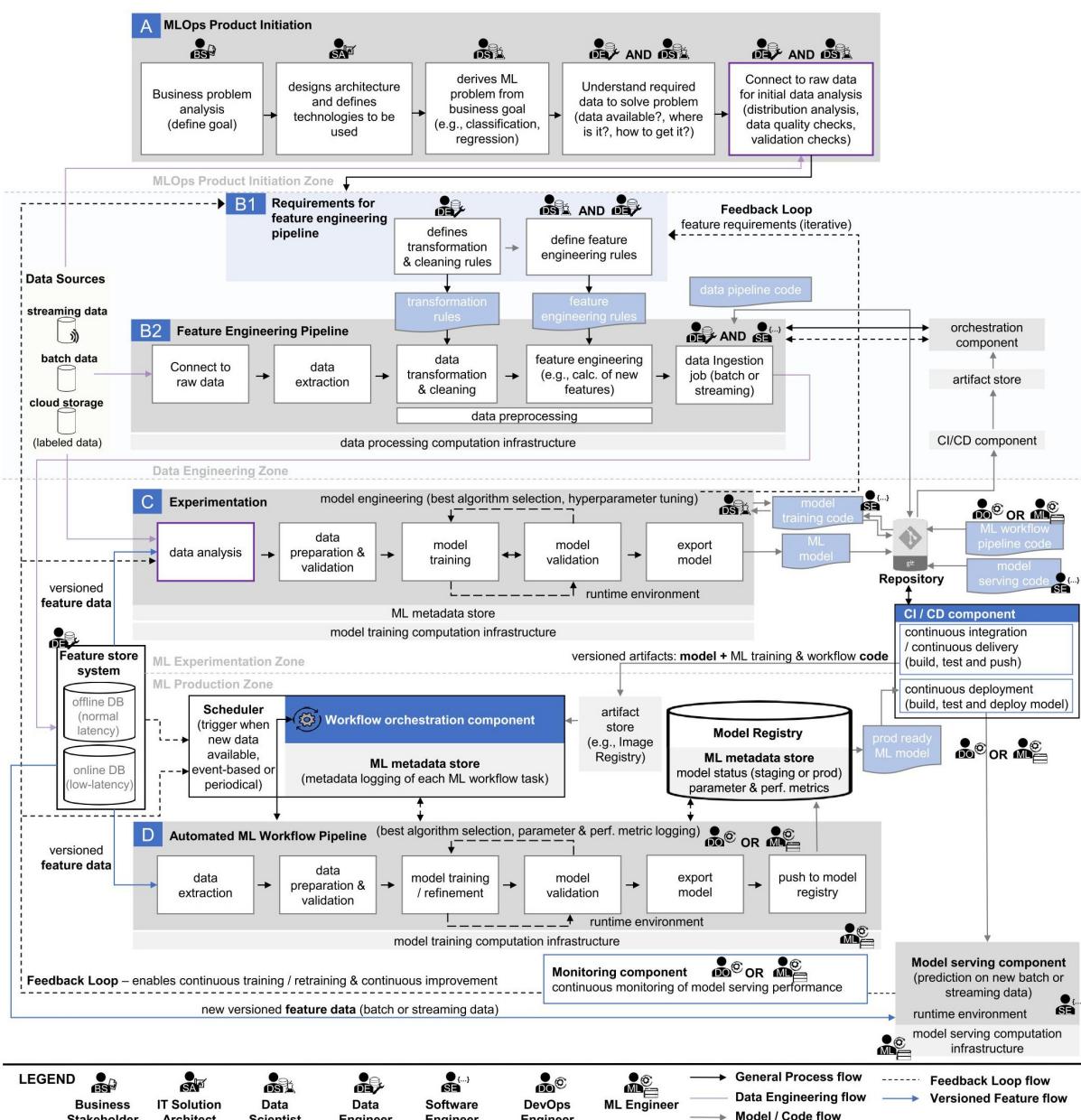
۳-۲-۳ نقش‌ها

در تولید یک پلتفرم MLOps، نقش‌های متعددی وجود دارد که همکاری آن‌ها برای طراحی، مدیریت، اتوماسیون و بهره‌برداری از سیستم‌های یادگیری ماشین در محیط تولید بسیار حیاتی است. در ابتدا سهام‌دار کسب‌وکار^{۲۳} وظیفه تعیین اهداف کسب‌وکار و استراتژی بازگشت سرمایه محصول یادگیری ماشین را بر عهده دارد. معمار، معماری سیستم را طراحی کرده و فناوری‌های مناسب را انتخاب می‌کند. دانشمند داده مسئله کسب‌وکار را به مسئله یادگیری ماشین ترجمه کرده و مدل‌ها را مهندسی می‌کند. مهندس داده خط لوله‌های داده و ویژگی را ایجاد و مدیریت می‌کند و داده‌ها را به درستی به سیستم‌های پایگاه داده و انبار ویژگی‌ها تزریق می‌کند. مهندس نرم‌افزار با استفاده از الگوهای طراحی، مسئله یادگیری ماشین را به یک محصول مهندسی شده تبدیل می‌کند. مهندس DevOps خودکارسازی CI/CD، هماهنگ‌سازی جریان کاری یادگیری ماشین و استقرار مدل در تولید را تضمین می‌کند. در نهایت، مهندس MLOps نقش ترکیبی از مهارت‌های چندگانه را دارد و زیرساخت یادگیری ماشین را ایجاد و مدیریت کرده، خط لوله‌های جریان کاری را خودکار می‌کند و مدل‌ها و زیرساخت را در تولید نظارت می‌کند. این نقش‌ها که در شکل ۳-۳ نشان داده شده است، با همکاری و هماهنگی نزدیک می‌توانند MLOps را به شکلی مؤثر و کارآمد پیاده‌سازی کنند، که نتیجه آن یک سیستم یادگیری ماشین پایدار و قابل اعتماد در محیط تولید خواهد بود. همکاری میان این نقش‌ها تضمین می‌کند که تمام جنبه‌های مربوط به توسعه، استقرار و نگهداری مدل‌های یادگیری ماشین به درستی مدیریت شود و به اهداف کسب‌وکار دست یابند.

Business Stakeholder^{۲۴}

۳-۳ معماری جامع

بر اساس اصول، اجزا و نقش‌های بیان شده، یک معماری جامع از یک پلتفرم MLOps طراحی شده است. این معماری که در شکل ۴-۳ نشان داده شده، جریان کارها و ترتیب وظایف در مراحل مختلف را ترسیم می‌کند. این معماری به‌گونه‌ای طراحی شده که کاربران می‌توانند مناسب‌ترین فناوری‌ها و چارچوب‌ها را بر اساس نیازهای خود انتخاب کنند. این انعطاف‌پذیری به کاربران این امکان را می‌دهد که پلتفرم MLOps را با استفاده از ترکیبی از ابزارهای متan باز، استفاده از سرویس‌های ابری یا



شکل ۴-۳: معماری جامع [۲] MLOps

رویکردهای ترکیبی پیاده‌سازی کنند. نرم‌افزارهای سازمانی و خدمات ابری اغلب از طریق API‌ها با ابزارهای متن‌باز یکپارچه می‌شوند و امکان ترکیب بی‌دردسر فناوری‌های مختلف را فراهم می‌کنند. این معماری یک معماری جامع بوده و هر سازمان می‌تواند بر حسب نیاز و هدف خود در حل مسئله یادگیری ماشین، این معماری را شخصی‌سازی کرده و از پیاده‌سازی قسمت‌هایی از آن صرف‌نظر کند [۲].

مرحله اولیه

در فرآیند شروع محصول MLOps، اولین مرحله با تحلیل کسب‌وکار آغاز می‌شود. کارشناس مربوط وظیفه دارد تا مشکلاتی را شناسایی کند که با استفاده از یادگیری ماشین قابل حل باشند. پس از شناسایی مشکل، معمار وارد عمل می‌شود. او طراحی معماری کلی سیستم یادگیری ماشین را تعریف کرده و پس از ارزیابی دقیق، تکنولوژی‌های مورد نیاز را انتخاب می‌کند. در مرحله بعدی، دانشمند داده باید از هدف کسب‌وکار، یک مسئله یادگیری ماشین استخراج کند. این مسئله بسته به ماهیت مشکل کسب‌وکار می‌تواند شامل طبقه‌بندی، رگرسیون یا دیگر روش‌های یادگیری ماشین باشد. برای این منظور، دانشمند داده با همکاری مهندس داده باید داده‌های موجود را تجزیه و تحلیل کند تا بهترین رویکرد برای حل مسئله را انتخاب کند. این مرحله نیازمند دانش عمیق از روش‌های مختلف یادگیری ماشین و توانایی تطبیق آن‌ها با نیازهای خاص پژوهه است. نکته مهم در این فرآیند، نیاز به داده‌های برچسب‌گذاری شده است که برای الگوریتم‌های نظارت شده ضروری هستند. در این معماری منابع داده از قبل دارای داده‌های برچسب‌گذاری شده بوده‌اند، زیرا فرآیند برچسب‌گذاری در مراحل قبلی انجام شده است.

خط لوله مهندسی ویژگی

در فرآیند توسعه مدل‌های یادگیری ماشین، مهندسی ویژگی‌ها^{۲۴} به عنوان یکی از گام‌های حیاتی شناخته می‌شود که مستلزم تعیین نیازمندی‌های اساسی و پیاده‌سازی خط لوله مهندسی ویژگی‌ها است. این مرحله شامل تعریف و پیاده‌سازی قواعد تبدیل و پاک‌سازی داده‌ها، و همچنین ایجاد ویژگی‌های جدید و پیشرفته بر اساس ویژگی‌های موجود است. ابتدا نیازمندی‌های مهندسی ویژگی‌ها توسط متخصص داده و مهندس داده تعریف می‌شوند. در این مرحله، قواعد تبدیل داده‌ها^{۲۵} مانند نرمال‌سازی و تجمعیح، و همچنین قواعد پاک‌سازی داده‌ها^{۲۶} تعیین می‌شوند تا داده‌ها به فرمت قابل استفاده تبدیل شوند. این قواعد اولیه، به صورت تکراری و بر اساس بازخوردهای حاصل از مراحل آزمایشی مهندسی مدل و یا از طریق نظارت بر

Feature Engineering^{۲۴}

Data Transformation^{۲۵}

Data Cleaning^{۲۶}

عملکرد مدل، تنظیم و بهبود می‌یابند.

با تعریف نیازمندی‌های اولیه، مهندس داده و مهندس نرم‌افزار اقدام به ساخت نمونه اولیه خط لوله تولید ویژگی‌ها می‌کنند. این خط لوله باید به صورت مداوم و بر اساس بازخوردهای دریافتی از مراحل مختلف، به روزرسانی و بهبود یابد. مراحل کلیدی پیاده‌سازی خط لوله تولید ویژگی‌ها به صورت زیر است:

- اتصال به داده‌های خام: اولین مرحله در پیاده‌سازی خط لوله تولید ویژگی‌ها، اتصال به منابع داده خام است. این داده‌ها می‌توانند از منابع مختلفی مانند داده‌های جریانی^{۲۷}، داده‌های دسته‌ای^{۲۸} یا داده‌های ذخیره‌شده در ابر^{۲۹} باشند. داده‌های جریانی به صورت پیوسته و بلاذرنگ دریافت می‌شوند، داده‌های دسته‌ای ثابت به صورت دوره‌ای و در حجم بالا جمع‌آوری و پردازش می‌شوند و داده‌های ذخیره‌شده در ابر از طریق سیستم‌های ابری مقیاس‌پذیر و انعطاف‌پذیر ذخیره می‌شوند. اتصال به منابع داده باید به‌گونه‌ای انجام شود که داده‌ها به راحتی قابل استخراج و پردازش باشند.
- استخراج داده‌ها: پس از اتصال به منابع داده، مرحله بعدی استخراج داده‌ها از این منابع است. این مرحله شامل خواندن داده‌ها از پایگاه‌های داده، فایل‌های CSV یا دیگر منابع داده است.
- پیش‌پردازش داده‌ها: در این مرحله، داده‌های استخراج شده برای تبدیل به فرم قابل استفاده، پیش‌پردازش می‌شوند. پیش‌پردازش شامل مراحل مختلفی مانند پاک‌سازی داده‌ها، مدیریت مقادیر مفقود، حذف نویز، و نرمال‌سازی مقادیر است. هدف اصلی این مرحله، آماده‌سازی داده‌ها به‌گونه‌ای است که بتوانند به عنوان ورودی‌های مدل یادگیری ماشین استفاده شوند.
- استخراج ویژگی‌های جدید و پیشرفت: یکی از مهم‌ترین مراحل در خط لوله تولید ویژگی‌ها، استخراج ویژگی‌های جدید و پیشرفت است. این ویژگی‌ها بر اساس ویژگی‌های موجود و با استفاده از تکنیک‌های مختلفی مانند ترکیب ویژگی‌ها، اعمال توابع ریاضی، و بهره‌گیری از روش‌های آماری ایجاد می‌شوند. این مرحله به مدل یادگیری ماشین کمک می‌کند تا الگوهای پیچیده‌تری را در داده‌ها شناسایی کند و دقت پیش‌بینی‌های خود را افزایش دهد.
- انتقال ویژگی‌ها به انبار ویژگی‌ها: در نهایت، داده‌های پردازش شده و ویژگی‌های محاسبه شده به انبار ویژگی‌ها وارد می‌شوند. این انبار می‌تواند شامل پایگاه‌های داده آنلاین یا آفلاین باشد. این بارگذاری باید به‌گونه‌ای انجام شود که دسترسی سریع و کارآمد به داده‌ها برای مراحل بعدی آموزش مدل فراهم شود.

Streaming Data^{۲۷}

batch data^{۲۸}

Cloud Storage^{۲۹}

در خط تولید ویژگی‌ها، مهندس نرم‌افزار به کمک مهندس داده کدهای مورد نیاز برای CI/CD و هماهنگ‌سازی را تعریف می‌کند تا وظایف خط تولید ویژگی‌ها به درستی هماهنگ شوند. این نقش شامل تنظیم منابع زیرساختی برای اطمینان از مقیاس‌پذیری و عملکرد بهینه خط تولید است. با این تنظیمات، خط تولید ویژگی‌ها می‌تواند به طور مداوم به روزرسانی شده و بر اساس بازخوردها بهبود یابد، که این امر بهبود عملکرد مدل‌های یادگیری ماشین را تضمین می‌کند.

برای پیاده‌سازی خطوط تولید ویژگی‌ها، از ابزارها و فناوری‌های مختلفی از جمله ابزارهای Apache ETL سنتی مانند Airflow استفاده می‌شوند. همچنین از Feast به همراه پایگاه داده‌های معروف مانند PostgreSQL و Redis نیز برای انبار ویژگی‌ها استفاده می‌شود.

بررسی و آزمایش

مرحله آزمایش مدل در فرآیند یادگیری ماشین یک بخش حیاتی است که بیشتر توسط دانشمند داده به همراه مهندس نرم‌افزار انجام می‌شود. قبل از شروع به کار، مهندس داده به همراه مهندس نرم‌افزار برای اطمینان از عملکرد درست ابزار و منابع، محیط و سخت‌افزار را پیکربندی می‌کنند. حال فرآیند آزمایش مدل شروع می‌شود:

- اتصال به انبار ویژگی: دانشمند داده به سیستم انبار ویژگی‌ها متصل می‌شود تا داده‌ها را برای تجزیه و تحلیل دریافت

کند. در صورت نیاز، داده خام نیز می‌تواند برای تحلیل‌های اولیه مورد استفاده قرار گیرد. اگر تغییراتی در داده‌ها لازم باشد، این تغییرات به تیم مهندسی داده گزارش می‌شود، که نتیجه آن می‌تواند منجر به تغییر قواعد تبدیل، پاکسازی داده‌ها و خط تولید ویژگی‌ها شود.

- آماده‌سازی و اعتبارسنجی داده‌ها: داده‌ها از سیستم انبار ویژگی‌ها جمع‌آوری و اعتبارسنجی می‌شوند. این مرحله شامل

آماده‌سازی داده‌ها و تقسیم آن‌ها به مجموعه‌های آموزش و تست و ارزیابی است تا مدل‌ها بتوانند به طور مؤثری آموزش داده شوند.

- آموزش و اعتبارسنجی مدل: در این مرحله، دانشمند داده الگوریتم‌های مختلف و پارامترهای آن‌ها را ارزیابی می‌کند تا

بهترین ترکیب را پیدا کند. آموزش مدل با استفاده از داده‌های آموزشی شروع می‌شود و مهندس نرم‌افزار در ایجاد کدهای آموزشی بهینه کمک می‌کند. مدل‌ها با استفاده از پارامترهای مختلف به صورت تعاملی آموزش و اعتبارسنجی می‌شوند. این فرآیند تکراری است و تا زمانی که مدل به عملکرد مطلوبی برسد، ادامه می‌یابد. هدف این مرحله شناسایی بهترین الگوریتم و پارامترهای بهینه است.

- استخراج مدل و ثبت کد: پس از شناسایی و انتخاب بهترین مدل، دانشمند داده مدل نهایی را استخراج کرده و کدهای مربوطه را در مخزن کد منبع قرار می‌دهد. این کدها شامل تمامی اسکریپت‌ها و مستنداتی است که برای تولید، آموزش و ارزیابی مدل استفاده شده‌اند. در همین زمان، مهندس DevOps یا مهندس یادگیری ماشین کدهای مربوط به خط لوله یادگیری ماشین را آماده و در مخزن قرار می‌دهد. این خط لوله شامل اسکریپت‌ها و تنظیماتی است که برای خودکارسازی فرآیندهای مختلف یادگیری ماشین مانند آموزش، ارزیابی و استقرار مدل مورد نیاز است. با انجام این کار، سیستم CI/CD به صورت خودکار تغییرات را تشخیص داده و فرآیند ساخت، آزمون و تحويل مدل را آغاز می‌کند. در مرحله ساخت، مؤلفه‌های مدل^{۳۰} و کدهای مرتبط ایجاد می‌شوند. در مرحله آزمون، صحت و عملکرد مدل بررسی می‌شود و در نهایت، در مرحله تحويل مدل نهایی به مخزن مؤلفه‌ها ارسال می‌شود تا برای استفاده در محیط عملیاتی آماده باشد.
- در مرحله آزمایش، ابزارهای مبتنی بر Notebook Jupyter [۴۰] به طور گسترده استفاده می‌شوند. این ابزارها به دانشمندان داده اجازه می‌دهند تا داده‌ها را آماده، مدل‌ها را آموزش، ارزیابی و بهینه‌سازی کنند. همچنین برای پیگیری و مدیریت آزمایش‌ها از ابزارهایی مانند MLflow و TensorBoard استفاده می‌شود.

خودکارسازی جریان کاری یادگیری ماشین

خودکارسازی جریان کاری یادگیری ماشین شامل مجموعه‌ای از فرآیندهای پیچیده و حیاتی است که توسط مهندس DevOps و مهندس یادگیری ماشین مدیریت می‌شود. این فرآیندها شامل مدیریت محیط‌های اجرایی و زیرساخت‌های لازم برای آموزش مدل‌ها است که از منابع سخت‌افزاری و فریم‌ورک‌های محاسباتی نظری کوبرنتیز استفاده می‌کنند. در این سیستم، یک مؤلفه ارکستراسیون وظایف مختلف را در جریان کاری خودکار یادگیری ماشین هماهنگ می‌کند. این مؤلفه وظایف را به محیط‌های مجزا (مانند کانتینرها) تخصیص داده و فراداده‌های هر وظیفه را در قالب لاغ‌ها و سایر اطلاعات جمع‌آوری می‌کند. مراحل اجرای این فرآیند که به قسمت قبل خیلی شباهت دارد به صورت زیر است:

- استخراج داده‌ها: اولین مرحله در این فرآیند، استخراج داده‌ها از سیستم‌های انبار ویژگی‌ها است. این داده‌ها می‌توانند از پایگاه‌های داده آنلاین یا آفلاین استخراج شوند. بسته به نیاز مورد استفاده، داده‌ها از منابع مختلفی استخراج شده و برای مراحل بعدی آماده می‌شوند.

- آماده‌سازی و اعتبارسنجی داده‌ها: در این مرحله، داده‌ها به صورت خودکار آماده‌سازی و اعتبارسنجی می‌شوند. همچنین، تقسیم‌بندی داده‌ها به مجموعه‌های آموزش و تست نیز به صورت خودکار انجام می‌گیرد. این فرآیند تضمین

می‌کند که داده‌های ورودی به مدل‌ها با کیفیت و قابل اعتماد باشند.

- آموزش مدل نهایی: پس از آماده‌سازی داده‌ها، مدل نهایی بر روی داده‌های جدید و نادیده آموزش داده می‌شود.

الگوریتم‌ها و ابرپارامترها بر اساس تنظیمات مراحل آزمایشی قبلی از پیش تعریف شده‌اند. در این مرحله، مدل آموزش داده شده و بهینه‌سازی می‌شود تا بهترین عملکرد ممکن را ارائه دهد.

- ارزیابی و تنظیم مدل: مدل آموزش دیده شده به صورت خودکار ارزیابی می‌شود و در صورت نیاز، ابرپارامترها تغییر می‌کنند. این فرآیند به صورت تکراری انجام می‌شود تا زمانی که معیارهای عملکرد نشان‌دهنده نتایج مطلوب باشند. این

تکرارها تا دستیابی به یک مدل با عملکرد بهینه ادامه می‌یابند.

- ثبت و ذخیره مدل: مدل نهایی آموزش دیده سپس ذخیره شده و به یک مخزن مدل منتقل می‌شود. این مخزن مدل،

مدل‌ها را به صورت کد یا کانتینر همراه با فایل‌های تنظیمات و محیط ذخیره می‌کند. این امر تضمین می‌کند که مدل‌ها به راحتی قابل دسترسی و استفاده مجدد باشند.

برای هر بار آموزش مدل، مخزن فراداده‌ها پارامترهای مورد نیاز برای آموزش مدل و معیارهای عملکرد حاصل را ثبت می‌کند. این شامل ثبت جزئیات هر دوره آموزش مدل، مانند تاریخ و زمان آموزش، مدت زمان، پارامترهای استفاده شده و معیارهای عملکرد مدل می‌باشد. همچنین نسخه و وضعیت مدل (مثلاً آماده برای تولید یا در حال توسعه) نیز ثبت می‌شود.

پس از انتقال مدل با عملکرد بالا از مرحله آزمایش به تولید، این مدل به طور خودکار به مهندس DevOps یا مهندس یادگیری ماشین برای استقرار مدل تحویل داده می‌شود. در این مرحله، ابزار مدیریت CI/CD مانند ArgoCD، خط لوله CD را اجرا می‌کند. مدل آماده و کدهای استقرار مدل که توسط مهندس نرم‌افزار تهیه شده‌اند، فرآخوانی می‌شوند. خط لوله CD وظیفه ساخت و آزمایش مدل و کدهای استقرار مدل برای استقرار در محیط عملیاتی را بر عهده دارد. برنامه‌های استقرار مدل اغلب با استفاده از یک کانتینر پیاده‌سازی و تنظیم می‌شوند و درخواست‌های پیش‌بینی را از طریق REST API پاسخ می‌دهند. هنگام استقرار یک برنامه یادگیری ماشین، استفاده از آزمایش B/A به عنوان یک استراتژی تست خوب توصیه می‌شود تا در یک سناریوی واقعی مشخص شود که کدام مدل بهتر عمل می‌کند.

مؤلفه نظارتی به صورت پیوسته عملکرد مدل و زیرساخت‌ها را پایش می‌کند. زمانی که یک آستانه خاص مانند تغییر در توزیع خروجی مدل تشخیص داده شود، اطلاعات از طریق حلقه بازخورد ارسال می‌شود. این حلقه امکان آموزش و بازآموزی

مداوم^{۳۱} و بهبود مستمر را فراهم می‌کند. اطلاعات از مؤلفه نظارتی مدل به چندین نقطه مانند مرحله آزمایش، مرحله تولید ویژگی و مهندسی داده منتقل می‌شود. بازخورد به مرحله آزمایش توسط دانشمند داده برای بهبود بیشتر مدل‌ها مورد استفاده قرار

می‌گیرد. بازخورد به مرحله تولید ویژگی نیز امکان تغییر در تولید ویژگی‌ها برای سیستم انبار ویژگی‌ها را فراهم می‌کند. تشخیص رانش داده به عنوان یک مکانیزم بازخورد نیز می‌تواند آموزش مستمر را فعال کند. رانش داده به تغییرات تدریجی یا ناگهانی در توزیع داده‌های ورودی مدل‌های یادگیری ماشین گفته می‌شود که می‌تواند باعث کاهش دقت و کارایی مدل‌ها شود. این تغییرات ممکن است به دلیل عوامل مختلفی مانند تغییر در رفتار کاربران، تغییر در شرایط محیطی، خرابی سنسورها و یا تغییرات سیستمی رخ دهنده. رانش داده به دو نوع اصلی تقسیم می‌شود:

رانش مفهوم^{۳۲}: تغییر در توزیع برچسب‌ها یا خروجی‌ها که نشان‌دهنده تغییر در الگوهای زیرین داده‌هاست.

رانش ویژگی^{۳۳}: تغییر در توزیع ویژگی‌ها یا ورودی‌های مدل که می‌تواند به دلیل تغییر در محیط یا منابع داده‌ها باشد. تشخیص رانش داده اهمیت زیادی دارد زیرا به مدل‌ها کمک می‌کند تا با تغییرات جدید سازگار شوند و از کاهش کارایی جلوگیری کنند. این تشخیص می‌تواند از طریق مقایسه توزیع‌های آماری قدیم و جدید داده‌ها و استفاده از الگوریتم‌های مختلف انجام شود. هنگامی که رانش داده تشخیص داده شود، مدل‌ها می‌توانند مجددآموزش داده شوند تا با شرایط جدید سازگار شوند و کارایی مطلوب خود را حفظ کنند.

تکنولوژی‌ها و ابزار برای پیاده‌سازی خط لوله خودکار یادگیری ماشین شامل Kubeflow، Apache Airflow، AWS SageMaker و Pipelines در زمینه تبلیغات آنلاین است [۴۱]. این شرکت از Airflow برای خودکارسازی فرآیند آموزش و استقرار مدل‌های یادگیری ماشین برای هدف‌گذاری و بهینه‌سازی تبلیغات استفاده می‌کند. در این خط لوله، داده‌های بزرگی از منابع مختلف مانند داده‌های کلیک‌استریم وب‌سایت، جمعیت‌شناسی کاربران و داده‌های عملکرد کمپین استخراج، تبدیل و بارگذاری می‌شوند. این داده‌ها سپس از یک سری مراحل پیش‌پردازش و مهندسی ویژگی عبور می‌کنند که به عنوان اپراتورهای Airflow پیاده‌سازی شده‌اند. در مرحله بعد، مدل‌های مختلف یادگیری ماشین بر روی داده‌های پردازش شده آموزش داده و ارزیابی می‌شوند. در نهایت، مدل با بهترین عملکرد برای تصمیم‌گیری‌های هدف‌گذاری تبلیغات بی‌درنگ به محیط تولید منتقل می‌شود. در این مثال، برای خودکارسازی کل فرآیند از جمله زمان‌بندی، نظارت و اجرای مجدد وظایف شکست‌خورده از Airflow استفاده می‌شود.

فصل ۱۴

طراحی پلتفرم MLOps

۱-۴ مقدمه

در این بخش بر اساس تعاریف و استانداردهایی که از یک پلتفرم MLOps می‌باشد، یک پلتفرم MLOps با استفاده از ابزارهای متن باز طراحی می‌شود. ابتدا با طراحی سیستم مدیریت پلتفرم شامل مباحثی نظیر مدیریت پیکربندی و فراهم‌سازی زیرساخت، مدیریت خطوط لوله CI/CD، مخازن کد منبع و مؤلفه‌ها شروع می‌کنیم و سپس به معماری خوشبی کوبرنتیز که قلب پلتفرم می‌باشد، می‌پردازیم. در این بخش، موضوعات مدیریت داده و مدل، شبکه، مدیریت کاربران و چند مستاجری، نظارت و استقرار مدل‌ها مورد بررسی قرار خواهد گرفت. در انتها نیز یک معماری جامع برای پلتفرم طراحی شده، داده خواهد شد.

۲-۴ سیستم مدیریت پلتفرم

۱-۲-۴ مدیریت پیکربندی و فراهم‌سازی زیرساخت

در دنیای فناوری اطلاعات، زیرساخت‌های تغییرناپذیر^۱ و تغییرپذیر^۲ دور رویکرد مهم در مدیریت و نگهداری سیستم‌ها هستند. زیرساخت‌های تغییرناپذیر به سیستم‌هایی اشاره دارند که پس از ایجاد، بدون تغییر باقی می‌مانند و در صورت نیاز به تغییر، سیستم‌های جدید جایگزین آن‌ها می‌شوند. این رویکرد با مزایایی همچون کاهش پیچیدگی‌های مدیریتی، افزایش قابلیت پیش‌بینی و کاهش ریسک‌های مرتبط با تغییرات ناخواسته همراه است [۱۱]. به کمک ابزارهایی مانند داکر و کوبرنتیز، پیاده‌سازی زیرساخت‌های تغییرناپذیر امکان‌پذیر است و از قابلیت مقیاس‌پذیری بالایی برخوردارند. سیستم‌های ابری غالباً از روش تغییرناپذیر استفاده کرده تا از مزایای آن بهره‌مند شوند. در مقابل، زیرساخت‌های تغییرپذیر به سیستم‌هایی اشاره دارند که

Immutable Infrastructure^۱
Mutable Infrastructure^۲

می‌توانند به‌طور پویا تغییر کنند و تنظیمات و پیکربندی‌های جدید را پذیرند. این رویکرد، انعطاف‌پذیری بیشتری را فراهم می‌کند و برای محیط‌هایی که نیاز به تغییرات مکرر دارند، مناسب‌تر است. با این حال، مدیریت تغییرات در زیرساخت‌های تغییرپذیر ممکن است چالش‌های بیشتری از جمله افزایش ریسک خطاهای نیاز به نظارت مداوم به همراه داشته باشد. انتخاب بین این دو رویکرد به نیازها و اولویت‌های سازمان بستگی دارد. در حالی که زیرساخت‌های تغییرپذیر برای محیط‌های تولید با نیاز به ثبات و قابلیت پیش‌بینی بالا مناسب‌ترند، زیرساخت‌های تغییرپذیر برای محیط‌های توسعه و آزمایش که نیاز به انعطاف‌پذیری دارند، کاربرد بیشتری دارند [۱۵]. این دو مفهوم به صورت مستقیم با مدیریت پیکربندی و فراهم‌سازی زیرساخت مرتبط هستند.

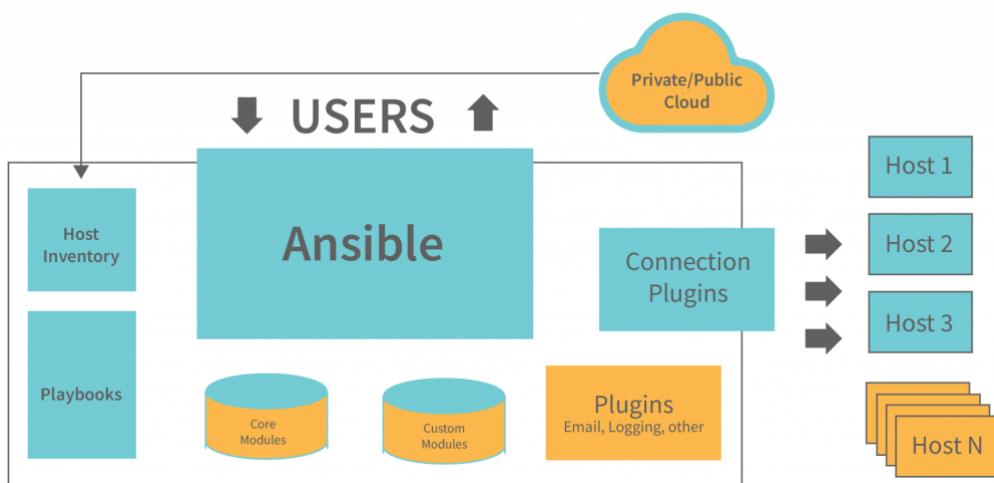
مدیریت پیکربندی

مدیریت پیکربندی فرآیندی است که بر روی نگهداری و کنترل پیکربندی سیستم‌ها و نرم‌افزارها تمرکز دارد. هدف اصلی این فرآیند، اطمینان از سازگاری و پایداری محیط‌های IT در طول زمان است. مدیریت پیکربندی شامل فعالیت‌هایی مانند نگهداری نسخه‌های مختلف نرم‌افزار، مستندسازی تغییرات و اطمینان از تطابق سیستم‌ها با استانداردهای تعیین شده می‌باشد. ابزارهای مدیریت پیکربندی مانند Ansible و Puppet به سازمان‌ها کمک می‌کنند تا فرآیندهای خودکارسازی پیکربندی را پیاده‌سازی کنند. این ابزارها از فایل‌های متنی (مانند Playbook‌ها در Ansible) برای تعریف وضعیت مطلوب سیستم‌ها استفاده می‌کنند. Ansible [۴۲] به دلیل سادگی و عدم نیاز به نصب عامل^۳ بر روی سیستم‌های مقصد، یکی از محبوب‌ترین ابزارهای مدیریت پیکربندی است. این ابزار از پروتکل SSH برای ارتباط با ماشین‌ها استفاده می‌کند و از زبان YAML برای نوشتن اسکریپت‌ها بهره می‌برد، که خوانایی و قابل فهم بودن آن را تضمین می‌کند. از ابزارهای مدیریت پیکربندی در زیرساخت‌های تغییرپذیر غالباً استفاده می‌شود.

فراهم‌سازی زیرساخت

فراهم‌سازی زیرساخت فرآیندی است که به راه‌اندازی و پیکربندی اولیه زیرساخت‌های IT اختصاص دارد. این فرآیند شامل ایجاد و مدیریت منابعی مانند سرورها، پایگاه داده‌ها، شبکه‌ها و سایر اجزای زیرساختی است. فراهم‌سازی زیرساخت به صورت سنتی فرآیندی دستی و زمان‌بر بود، اما با ظهور ابزارهای نوین این فرآیند بهشدت خودکار و ساده شده است. ابزارهای متن‌بازی مانند Terraform برای این کار استفاده می‌شوند. این ابزارها به کاربران امکان می‌دهند تا زیرساخت‌های خود را

Agent^۳

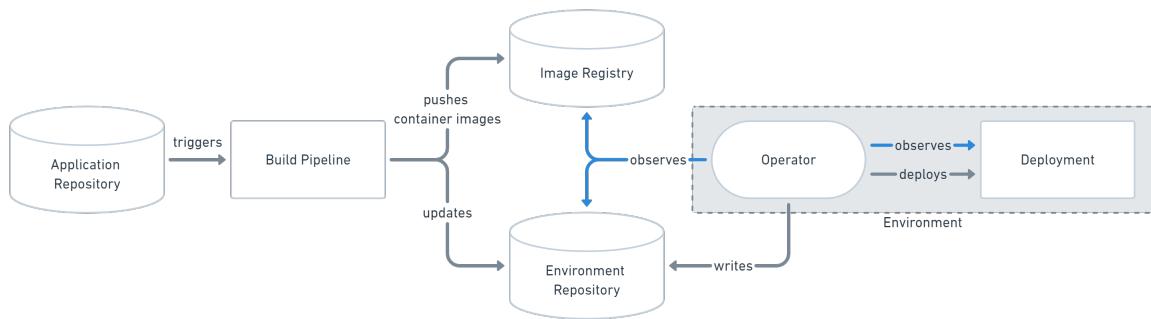


شکل ۱-۴: معماری Ansible

به صورت کد^۴ تعریف کنند [۱۰]. این رویکرد به ایجاد و مدیریت منابع زیرساختی به صورت قابل تکرار و پایدار کمک می‌کند. Terraform، یکی از پرکاربردترین ابزارهای فراهم‌سازی زیرساخت، از زبان HCL برای تعریف زیرساخت‌ها استفاده می‌کند. یکی از ویژگی‌های برجسته Terraform مدیریت وابستگی‌ها بین منابع است که امکان بازگشت^۵ به وضعیت‌های قبلی را نیز فراهم می‌کند [۴۳]. از این ابزارها غالباً برای زیرساخت‌های تغییرناپذیر استفاده می‌شود.

در زیرساخت طراحی شده که از OpenStack برای ساخت و مدیریت ماشین‌های مجازی استفاده می‌شود، استفاده از Ansible به عنوان ابزار مدیریت پیکربندی به دلایل متعددی مناسب است. اولاً، Ansible با استفاده از Playbook‌های YAML امکان خودکارسازی مراحل پیکربندی را فراهم می‌کند، از جمله نصب نرم‌افزارهای مورد نیاز، تنظیمات شبکه و پیکربندی سرویس‌ها. این ویژگی باعث می‌شود که پیکربندی‌ها به صورت دقیق و بدون خطا انجام شود. ثانیاً، Ansible بدون نیاز به نصب عامل بر روی ماشین‌های مجازی کار می‌کند و از پروتکل SSH برای ارتباط استفاده می‌کند که این امر فرآیند پیکربندی را ساده‌تر و سریع‌تر می‌سازد. همچنین، تمامی مراحل پیکربندی به صورت کد تعریف می‌شوند که امکان اجرای مجدد و دقیق همان تنظیمات را بر روی ماشین‌های جدید فراهم می‌کند. به علاوه، Ansible دارای ماژول‌های متعددی برای تعامل با OpenStack است که می‌تواند فرآیند ایجاد و مدیریت ماشین‌های مجازی را بهینه‌تر کند. پس از پیکربندی اولیه VM‌ها، Ansible می‌تواند خوشکوبرنتیز را به صورت خودکار راه‌اندازی و پیکربندی کند. این شامل نصب ابزارهای مورد نیاز، تنظیمات شبکه و پیکربندی سرویس‌های کوبرنتیز است.

Infrastructure as Code^۴
Rollback^۵



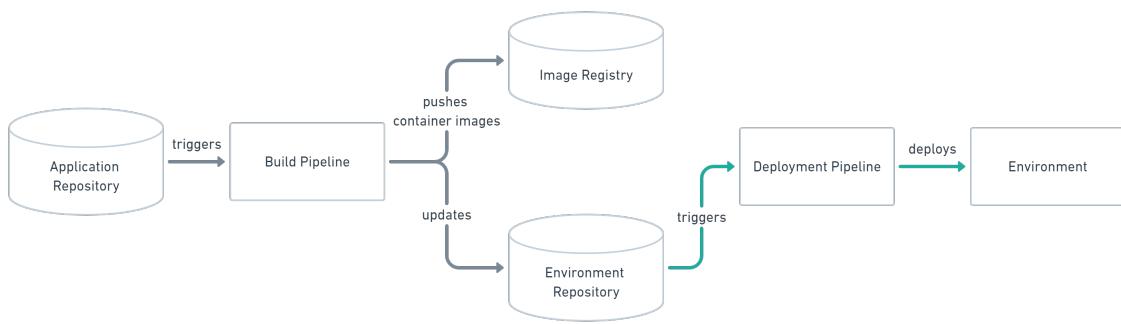
شکل ۲-۴: استقرار مبتنی بر Pull

۲-۲-۴ خط لوله CI/CD

با افزایش استفاده از سیستم‌های ابری و رویکرد تغییرنایذیر، مفهومی با عنوان GitOps معرفی شد که از گیت برای مدیریت زیرساخت و پیکربندی بهره می‌برد. GitOps در سال ۲۰۱۷ توسط Weaveworks معرفی شد که رویکردهی مدرن برای پیاده‌سازی خط لوله CD بر روی سیستم‌های ابری است. در حالی که ابزارهای سنتی تحویل مداوم عمدتاً از مدل Push استفاده می‌کنند، GitOps مدل Pull را معرفی می‌کند که به خصوص با کاتینرها و پیکربندی‌های اعلامی به خوبی کار می‌کند و آن را به یک روند محبوب در اکوسیستم بومی ابر تبدیل کرده است [۱۴]. از معروف‌ترین ابزار کلیدی که فرآیندهای GitOps را تسهیل می‌کند می‌توان به ArgoCD اشاره کرد.

همان‌طور که گفته شد دو رویکرد در پیاده‌سازی خط لوله CD وجود دارد. در مدل Pull (شکل ۲-۴) مانند GitOps، توسعه‌دهندگان حالت مطلوب را در مخزن گیت قرار می‌دهند. ابزاری نظیر ArgoCD در محیط تولید به صورت خودکار این تغییرات را شناسایی کرده و اعمال می‌کند. این مدل امنیت را افزایش می‌دهد زیرا نیازی به اعتبارنامه‌های دسترسی مستقیم برای توسعه‌دهندگان نیست. همچنین این مدل مشکل استقرارهای مبتنی بر Push را حل می‌کند، که در آن محیط تنها زمانی به روز می‌شود که مخزن محیط به روز شود. در مقابل، در مدل Push (شکل ۳-۴)، استقرار در محیط تولید شامل خطوط لوله CI/CD با اسکریپت‌هایی است که با هر تغییر در گیت فعال می‌شوند. این اسکریپت‌ها معمولاً ساخت، تست و در نهایت استقرار برنامه‌ها یا تنظیم پیکربندی‌های جدید در محیط تولید را با استفاده از ابزارهای خط فرمان و اعتبارنامه‌های ارائه شده انجام می‌دهند. این مدل کنترل دقیق‌تر بر فرآیند استقرار، اعمال سریع تغییرات، انعطاف‌پذیری بالا در مدیریت سناریوهای پیچیده و پشتیبانی بهتر از تغییرات جزئی را فراهم می‌کند، که در محیط‌های متنوع و پویا بسیار مفید است [۱۴].

به منظور پیاده‌سازی یک ابزار متن‌باز برای مدیریت خط لوله‌های CI/CD که برای سیستم‌های ابری نیز مناسب باشد، رویکرد تغییرنایذیر به همراه استراتژی استقرار مبتنی بر Push انتخاب شده است. انتخاب رویکرد تغییرنایذیر به دلیل نیاز به



شکل ۳-۴: استقرار مبتنی بر Push

انعطاف‌پذیری بیشتر در محیط‌هایی که تغییرات مکرر و بهروزرسانی‌های سریع دارند، انجام شده است. زیرساخت‌های تغییرپذیر به ما امکان می‌دهند تا به سرعت به تغییرات نیازمندی‌ها پاسخ دهیم و تنظیمات و پیکربندی‌های جدید را به راحتی اعمال کنیم. این ویژگی در محیط‌های توسعه و آزمایش بسیار حیاتی است، زیرا تغییرات مداوم و آزمایش‌های متعدد بخشی از فرآیند توسعه نرم‌افزار هستند. همچنین روش Push نیز به دلیل سادگی و کارایی در اعمال بهروزرسانی‌ها انتخاب شده است. با استفاده از این روش، می‌توانیم به روزرسانی‌ها را مستقیماً به سرورها ارسال کنیم و اطمینان حاصل کنیم که تمام سیستم‌ها به سرعت و بدون نیاز به مداخله دستی به روز می‌شوند. این رویکرد همچنین به کاهش زمان مورد نیاز برای انتشار تغییرات کمک می‌کند. در این راستا، Jenkins به عنوان ابزار پیاده‌سازی و مدیریت خط لوله CI/CD انتخاب شده است. جنکینز به دلیل متن‌باز بودن و دارا بودن تعداد زیادی پلاگین، انعطاف‌پذیری بسیار بالایی دارد و می‌تواند با انواع سیستم‌های ابری و رویکردهای زیرساختی سازگار شود. جنکینز همچنین با Ansible که به عنوان ابزار مدیریت پیکربندی انتخاب شد، به خوبی سازگار است. این ترکیب به ما اجازه می‌دهد تا پیکربندی‌های پیچیده را به سادگی مدیریت کنیم و اطمینان حاصل کنیم که تمام زیرساخت‌ها به صورت هماهنگ عمل می‌کنند.

۴-۲-۳ مخزن کد منبع

مخزن کد منبع^۶ یک سیستم ذخیره‌سازی و مدیریت کد است که به توسعه‌دهندگان این امکان را می‌دهد تا به صورت مشترک و هماهنگ بر روی پروژه‌های نرم‌افزاری کار کنند. این مخازن ابزارهای متعددی را برای تسهیل و بهبود فرآیند توسعه نرم‌افزار فراهم می‌کنند. یکی از اصلی‌ترین ویژگی‌های مخزن کد منبع، کنترل نسخه است که به توسعه‌دهندگان این امکان را می‌دهد تا تغییرات کد را پیگیری کرده و به نسخه‌های قبلی بازگردند. این ابزارها با ثبت تاریخچه تغییرات و شاخه‌بندی، امکان مدیریت همزمان چندین ویژگی یا رفع اشکال را فراهم می‌کنند بدون اینکه تغییرات یکدیگر را تحت تأثیر قرار دهند.

رایج‌ترین و پرکاربردترین این مخازن، گیت است که با ابزارهای مختلفی مانند GitHub، GitLab و Bitbucket می‌باشد. از آنجاکه یکی از شرط‌های پیاده‌سازی پلتفرم، متن‌باز بودن ابزارهای آن می‌باشد، GitLab برای مدیریت مخزن کد منبع استفاده شده است. در این طراحی Jenkins به عنوان مخزن کد متصل شده و هر تغییر در کد منبع باعث اجرا شدن یک خط لوله CI/CD مشخص توسط Jenkins می‌گردد.

۴-۲-۴ مخزن مؤلفه‌ها

یک مخزن مؤلفه^۷ یک سیستم متمرکز برای ذخیره‌سازی، مدیریت و انتشار مؤلفه‌های نرم‌افزاری است. این مؤلفه‌ها شامل هر نوع فایل باینری، کتابخانه، ماژول، پکیج، پلاگین یا حتی مستنداتی می‌شود که در طول چرخه عمر توسعه نرم‌افزار تولید می‌شود. هدف اصلی این مخازن این است که به تیم‌های توسعه اجازه دهد تا به راحتی نسخه‌های مختلفی از مؤلفه‌ها را مدیریت و به اشتراک بگذارند، فرآیندهای ساخت و انتشار را ساده کنند و وابستگی‌ها را طور مؤثرتری مدیریت کنند. همچنین، از انتشار مؤلفه‌هایی که هنوز تست نشده‌اند یا از نظر امنیتی مشکلاتی دارند جلوگیری می‌کنند. یکی از ابزارهای محبوب و متن‌باز برای مدیریت مخازن مؤلفه‌ها، Nexus است. از فرمتهای مختلف مؤلفه‌ها مانند Helm، apt، PyPI و Docker پشتیبانی می‌کند، که این امر آن را به یک ابزار چندمنظوره برای انواع پروژه‌های نرم‌افزاری تبدیل می‌کند. مخازن مؤلفه موردنیاز برای پیاده‌سازی پلتفرم در ۴ فرمت PyPI، apt، raw و Docker می‌باشد.

مخازن APT

یک سیستم مدیریت بسته در سیستم عامل‌های مبتنی بر دیبان است که به کاربران اجازه می‌دهد تا بسته‌های نرم‌افزاری را به راحتی نصب، به روزرسانی و حذف کنند. از آنجاکه بسته‌های استفاده شده در سرورها غالباً یکسان می‌باشد، به منظور افزایش سرعت پیاده‌سازی و اعمال تغییرات و پیکربندی تمامی بسته‌های مورد استفاده و نصب نشده در محیط، در Nexus ذخیره خواهد شد. این امر با استفاده از مخازنی از نوع Proxy انجام خواهد شد. در پیاده‌سازی سیستم علاوه بر بسته‌های موجود در APT رسمی Ubuntu، از مخازن Containerd و Kubernetes نیز برای نصب و پیاده‌سازی کوبرتیز نیز استفاده شده است. علاوه بر این، یک مخزن هم برای نصب Nvidia Driver و Nvidia CUDA Toolkit برای استفاده از واحد پردازنده گرافیکی ایجاد خواهد شد.

مخزن PyPI

یک مخزن عمومی برای بسته‌های نرم افزاری پایتون است که به توسعه‌دهندگان اجازه می‌دهد تا کتابخانه‌ها و ابزارهای خود را منتشر، به روزرسانی و مدیریت کنند. کاربران می‌توانند این بسته‌ها را به راحتی با استفاده از ابزار pip نصب کنند. همانند APT به منظور ذخیره‌سازی تمامی بسته‌های استفاده شده در محیط تولید ساخته شده‌اند. این امر باعث افزایش سرعت در نصب مجدد بسته‌ها و پایداری سیستم در زمان‌های قطعی یا خرابی مخازن رسمی می‌شود.

مخزن Docker

یک سرویس برای Docker Images است که به توسعه‌دهندگان اجازه می‌دهد تا تصاویر خود را ذخیره، مدیریت و به اشتراک بگذارند. با توجه به تحریم استفاده از DockerHub در ایران و همچنین کند بودن راههای جایگزین در گرفتن تصاویر موردنظر از مخازن رسمی این مخزن به وجود آمده که به مخزن رسمی docker پردازی شده است. علاوه بر این تصاویر لازم برای پیاده‌سازی و پیکربندی محیط که با خط لوله CI/CD ساخته شده‌اند نیز برای استفاده مجدد در این مخازن قرار می‌گیرند.

مخزن raw

برای ذخیره‌سازی و مدیریت فایل‌ها و داده‌هایی استفاده می‌شود که فرمت خاصی ندارند. این نوع مخزن به توسعه‌دهندگان اجازه می‌دهد تا انواع مختلف فایل‌ها، مانند اسکریپت‌ها، تصاویر، و مستندات را بدون نیاز به ساختاردهی خاصی نگهداری کنند.

۳-۴ معماری خوش کوبرنتیز

در طراحی یک پلتفرم MLOps جامع و کارآمد که تمامی ابزارهای مورد نیاز را در بر می‌گیرد، هدف اصلی ایجاد یک بستر یکپارچه، مقیاس‌پذیر و انعطاف‌پذیر برای مدیریت چرخه حیات مدل‌های یادگیری ماشین است. این پلتفرم شامل مجموعه‌ای از ابزارها و تکنولوژی‌های متن‌باز است که همگی روی خوش کوبرنتیز مستقر می‌شوند. استفاده از کوبرنتیز در پلتفرم‌های MLOps به دلیل قابلیت‌های منحصر به فرد آن در مقیاس‌پذیری و مدیریت خودکار منابع است. کوبرنتیز امکان استقرار مدل‌های یادگیری ماشین در کانتینرها را به صورت پویا و قابل اطمینان فراهم می‌کند، که این امر منجر به بهبود فرایندهای استقرار و به روزرسانی مدل‌ها می‌شود. همچنین، کوبرنتیز با ارائه قابلیت‌های مانیتورینگ و لاگینگ پیشرفته، به تشخیص و رفع سریع مشکلات کمک می‌کند و با ابزارهایی مانند Kubeflow، مدیریت چرخه عمر مدل‌ها را تسهیل می‌کند. در ادامه، با الهام از

اصول، اجزا و معماری جامع بیان شده در فصل سوم، معماری اصلی پلتفرم MLOps را که شامل تمام ابزارهایی که بر روی خوشکوبرنتیز پیاده‌سازی می‌شوند، طراحی کرده و برای هر بخش یک ابزار مناسب معرفی می‌کنیم.

۱-۳-۴ مدیریت داده و مدل

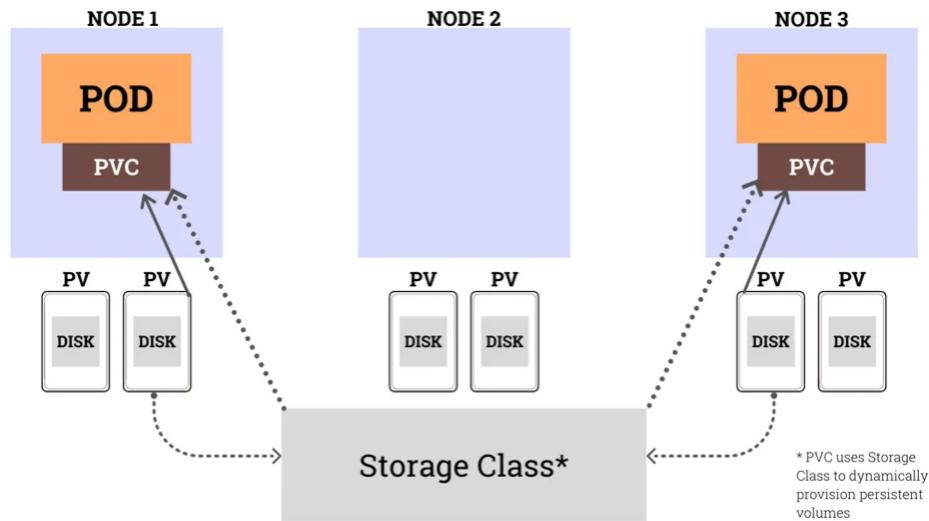
داده‌ها در پلتفرم MLOps نقش مهمی دارند. دانشمندان داده به منظور پیاده‌سازی یک مدل یادگیری ماشین نیاز به آموزش این مدل‌ها با استفاده از داده‌های از پیش آماده دارند. این داده‌ها می‌توانند به صورت متن، تصویر یا صوت باشد. لذا وجود یک محل ذخیره‌سازی داده برای نگهداری داده‌ها در این پلتفرم اساسی است. در کوبرنتیز، چندین روش و نوع ذخیره‌سازی داده وجود دارد که یکی از مهم‌ترین آن‌ها PVC می‌باشد که از PV برای ذخیره‌سازی داده‌ها استفاده می‌کند.

حجم پایدار و فراهم‌سازهای پویا

در کوبرنتیز PV به عنوان منابع ذخیره‌سازی مستقل از چرخه حیات پادها عمل می‌کنند، به این معنا که داده‌ها پس از حذف یا بازسازی پادها همچنان حفظ می‌شوند. PV‌ها توسط مدیر خوشکوبرنتیز به صورت ایستا یا پویا ایجاد می‌شوند و می‌توانند به PVC متصل شوند. PVC یک درخواست برای ذخیره‌سازی است که توسط کاربران ایجاد می‌شود و پس از تخصیص، به یک PV مرتبط می‌شود. این مکانیزم باعث می‌شود که مدیریت ذخیره‌سازی در خوشکوبرنتیز ساده‌تر و کارآمدتر شود. PV‌ها می‌توانند از انواع مختلف ذخیره‌سازی مانند دیسک‌های محلی، شبکه‌های ذخیره‌سازی^۸ یا راه حل‌های ابری استفاده کنند. به علاوه، استفاده از PV و PVC امکان استفاده مجدد از منابع ذخیره‌سازی را بدون نیاز به تنظیمات دستی پیچیده، برای پادهای مختلف فراهم می‌کند [۴۴، ۴۵].

یکی از قابلیت‌های مهم کوبرنتیز، فراهم‌سازی پویا^۹ است که به طور خودکار PV‌ها را براساس نیازهای PVC‌ها و کلاس‌های ذخیره‌سازی^{۱۰} ایجاد می‌کند. این ویژگی، نیاز به ایجاد و مدیریت دستی PV را توسط مدیران خوشکوبرنتیز از بین می‌برد و مدیریت ذخیره‌سازی را ساده‌تر می‌کند. با تعریف کلاس‌های ذخیره‌سازی، می‌توان انواع مختلفی از ذخیره‌سازی را با ویژگی‌های مورد نظر فراهم کرد. هنگامی که یک درخواست ذخیره‌سازی PVC با کلاس ذخیره‌سازی مشخص ایجاد می‌شود، کوبرنتیز به طور خودکار یک PV که متناسب با درخواست است را ایجاد و به درخواست متصل می‌کند [۴۴]. این فرآیند خودکار، نه تنها کارایی و بهره‌وری را افزایش می‌دهد بلکه اطمینان می‌دهد که منابع ذخیره‌سازی به صورت بهینه و کارآمد تخصیص داده می‌شوند. ابزار متن‌بازی که در این پلتفرم برای مدیریت و ساخت PV‌ها استفاده شده است، OpenEBS می‌باشد. با استفاده از

Network File System^۸
Dynamic Provisioning^۹
Storage Class^{۱۰}



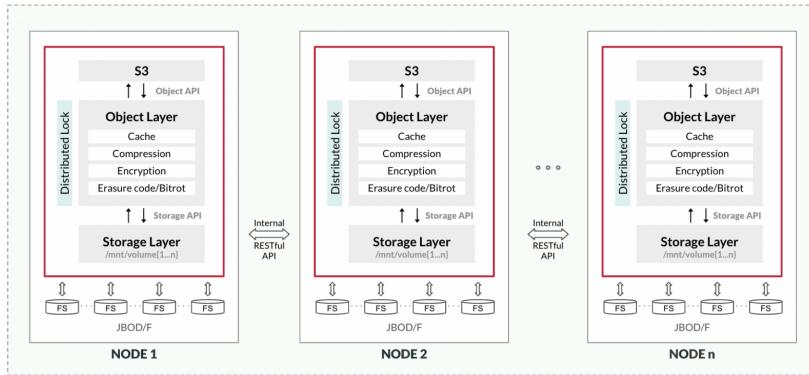
شکل ۴-۴: نحوه کار PVC و PV در خوشة کوبرتینز

این ابزار ما PV‌های مورد نیاز برای ذخیره‌سازی داده‌های پایگاه داده و ذخیره‌سازی شیء مانند MinIO را ایجاد می‌کنیم.

ذخیره سازی داده

برای پلتفرم‌های MLOps که نیاز به ذخیره‌سازی حجم زیادی از داده‌ها دارند، استفاده از فضای ذخیره‌سازی ضروری است. ذخیره‌سازی شیء^{۱۱} یک مدل ذخیره‌سازی داده است که برای مدیریت و دسترسی به داده‌های بدون ساختار مانند فایل‌های متند، تصویری و صوتی استفاده می‌شود. در این مدل، داده‌ها به عنوان اشیاء با شناسه منحصر به فرد به همراه فراداده‌ها ذخیره می‌شوند. مزایای اصلی ذخیره‌سازی شیء شامل مقیاس‌پذیری بالا، هزینه کمتر و انعطاف‌پذیری در مدیریت داده‌ها است. این سیستم‌ها به راحتی می‌توانند به میلیارد داده شیء افزایش یابند و برای استفاده در محیط‌های ابری، پشتیبان‌گیری و آرشیو داده‌ها مناسب هستند. همچنین، مدیریت ساده‌تر فراداده‌ها امکان جستجو و دسترسی سریع‌تر به داده‌ها را فراهم می‌کند. MinIO [۴۶] یک نرم‌افزار متن‌باز است که به منظور ارائه سرویس ذخیره‌سازی داده طراحی شده است. این نرم‌افزار به دلیل عملکرد بالا و پایداری که دارد، در محیط‌های مختلف از جمله سیستم‌های ابری به کار گرفته می‌شود. MinIO به عنوان جایگزینی برای Amazon S3 مطرح شده و سازگاری کامل با API‌های S3 را دارد، که این امر مهاجرت بین این دو سیستم را تسهیل می‌کند. معماری MinIO به صورت توزیع‌شده طراحی شده است که این امکان را فراهم می‌کند تا داده‌ها به صورت افقی مقیاس‌پذیر باشند. این معماری به گونه‌ای است که می‌توان با اضافه کردن گره‌های جدید به خوشة، ظرفیت و عملکرد سیستم را به سادگی افزایش داد. در خوشة MinIO، داده‌ها به صورت خودکار توزیع و تکرار^{۱۲} می‌شوند تا از دسترس پذیری بالا و تحمل

Object Storage^{۱۱}
Replication^{۱۲}



شکل ۵-۴: معماری MinIO

خطا اطمینان حاصل شود. یکی از مهم‌ترین تکنیک‌هایی که MinIO برای ذخیره‌سازی داده‌ها به کار می‌گیرد، کدگذاری پاکسازی^{۱۳} است. این تکنیک به MinIO اجازه می‌دهد تا داده‌ها را به بلوک‌های کوچک‌تر تقسیم کرده و آن‌ها را به صورت توزیع شده در چندین گره ذخیره کند. در صورت خرابی یک یا چند گره، MinIO می‌تواند داده‌ها را از بلوک‌های باقی‌مانده بازیابی کند، بدون اینکه داده‌ای از دست برود. این روش نه تنها فضای ذخیره‌سازی را بهینه می‌کند بلکه تحمل خطای سیستم را نیز افزایش می‌دهد. این ویژگی‌ها MinIO را به انتخابی مناسب برای ذخیره‌سازی داده‌های حجمی و پشتیبان‌گیری از آن تبدیل کرده‌اند. در کنار این‌ها یکی از ویژگی‌های مهم MinIO، سادگی و کاربرپسندی آن است. نصب و راهاندازی این نرم‌افزار بسیار ساده بوده و با چند فرمان ساده قابل انجام است. رابط کاربری وب و خط فرمان (mc) نیز به کاربران اجازه می‌دهند تا مدیریت و مانیتورینگ سرورها و داده‌ها را به سادگی انجام دهند.

در معماری طراحی شده برای این پلتفرم، داده‌ها ابتدا در MinIO و در باکت‌های مجزا ذخیره می‌شوند. دانشمندان داده به همراه مهندسان داده با استفاده از کتابخانه‌های پایتون از این داده‌ها استفاده کرده و پیش‌پردازش‌های موردنظر را روی داده‌ها انجام داده و ویژگی‌های موردنظر را استخراج می‌کنند. در نهایت این ویژگی‌ها را می‌توانند در همان MinIO و یا در پایگاه‌داده‌هایی که به همین منظور طراحی شده ذخیره نمایند.

پایگاه داده

به منظور ذخیره‌سازی داده‌های ساختاریافته^{۱۴} مانند داده‌های جدولی و یا ویژگی‌های به دست آمده در خط لوله مهندسی ویژگی می‌توان از پایگاه‌های داده استفاده نمود. در این راستا، از PostgreSQL و Redis به عنوان انباره‌های داده آنلاین و آفلاین استفاده شده است.

Erasuer Coding^{۱۳}
Structured^{۱۴}

PostgreSQL به عنوان انباره داده آفلاین مورد استفاده قرار می‌گیرد. این سیستم مدیریت پایگاه داده رابطه‌ای، به دلیل پشتیبانی از تراکنش‌های ACID، قابلیت اطمینان بالا و توانایی پشتیبانی از انواع داده‌های پیچیده، انتخاب مناسبی برای ذخیره‌سازی داده‌های آموزشی و ویژگی‌های به دست آمده از آن، لاگ‌ها و اطلاعات تحلیلی است. PostgreSQL با قابلیت‌های تحلیلی قوی و مقیاس‌پذیری مناسب، امکان تجزیه و تحلیل عمیق داده‌ها و مدیریت متاداده‌های مدل‌ها را فراهم می‌کند.

در مقابل، Redis یک انباره داده در حافظه^{۱۵} است که برای کاربردهای آنلاین در معماری MLOps بسیار مناسب است. Redis به دلیل سرعت بالای خواندن و نوشتن داده‌ها و پشتیبانی از ساختارهای داده متنوع، برای ذخیره‌سازی نتایج پیش‌بینی مدل‌ها، کش کردن داده‌های موقت و مدیریت صفات و تراکنش‌های سریع استفاده می‌شود. این امر به بهبود کارایی و کاهش زمان پاسخ‌دهی سیستم کمک می‌کند.

ترکیب PostgreSQL و Redis در این معماری، یک راهکار جامع و کارآمد برای مدیریت داده‌ها فراهم می‌کند. PostgreSQL با امکانات پیشرفته‌اش به عنوان انباره داده آفلاین، و Redis با سرعت بالایش به عنوان انباره داده آنلاین می‌تواند نیازهای مختلف سیستم را فراهم سازد.

بانک مدل و انبار فراداده

بانک مدل یکی از ابزارهای بسیار مهم در مدیریت مدل‌های یادگیری ماشین است که به تیم‌ها کمک می‌کند تا مدل‌های خود را به صورت سازمان‌یافته ذخیره، مدیریت و ردیابی کنند. همچنین اطلاعات مربوط به هر مدل را از جمله نسخه، تاریخ آخرین آموزش، معیارهای ارزیابی و مستندات مربوطه را نگهداری می‌کند. این امر به تیم‌ها کمک می‌کند تا با استفاده از نسخه‌های مختلف مدل‌ها، آزمایش‌های مختلفی انجام دهند و بهترین مدل را انتخاب کنند. همچنین به هنگام بروز مشکل در مدل‌های جدید، می‌توان از مدل‌های قابل قبول برای محیط عملیاتی استفاده کرد. انبار فراداده یادگیری ماشین نیز برای پیگیری و ذخیره‌سازی اطلاعات مربوط به هر مرحله از جریان کاری یادگیری ماشین استفاده می‌شوند. فراداده‌ها می‌توانند شامل جزئیاتی نظیر تاریخ و زمان آموزش مدل، مدت زمان هر مرحله از آموزش، پارامترهای استفاده شده، معیارهای عملکرد مدل و سلسله‌مراتب مدل (مثل داده‌ها و کدهای استفاده شده) باشند.

از ابزارهای معروف متن‌باز در این بخش می‌توان به MLflow اشاره کرد. MLflow یک پلتفرم مدیریت چرخه عمر یادگیری ماشین است که توسط Databricks توسعه داده شده است. این ابزار به تیم‌های یادگیری ماشین کمک می‌کند تا فرآیند توسعه، آزمایش و مدیریت مدل‌های خود را بهبود بخشدند. این ابزار شامل چهار جزء اصلی است: مدیریت آزمایش^{۱۶} که امکان

In-memory^{۱۵}
Experiment Management^{۱۶}

ردیابی و مقایسه آزمایش‌های مختلف را فراهم می‌کند، ردیابی^{۱۷} که به ثبت و مدیریت پارامترها، معیارهای عملکرد و مدل‌های ساخته شده کمک می‌کند، بسته‌بندی^{۱۸} که مدل‌های آموزش دیده را به صورت استاندارد و قابل اجرا در محیط‌های مختلف بسته‌بندی و ذخیره‌سازی می‌کند، و استقرار که اجرای مدل‌ها را در محیط‌های مختلف مانند داکر و کوبرنتیز ساده‌تر می‌کند.

با توجه به کاهش پیچیدگی در طراحی می‌توان از ابزارهای استفاده شده در بخش‌های دیگر برای پیاده‌سازی این بخش استفاده کرد. از آنجاکه MinIO توانایی ذخیره‌سازی هر مؤلفه‌ای را به همراه فراداده‌های آن دارد، از این ابزار برای Packaging استفاده شده است. سایر بخش‌ها نیز به دلیل یکپارچگی با اجزا دیگر در پلتفرم طراحی شده با استفاده از Kubeflow پیاده‌سازی شده است.

۲-۳-۴ شبکه

بسیاری از برنامه‌های مدرن با استفاده از معماری میکروسرویس توزیع شده ساخته می‌شوند که باعث می‌شود هر سرویس ساده و دارای مسئولیت مشخص باشد. هر میکروسرویس API‌های خود را تعریف کرده و سرویس‌ها برای پاسخ‌گویی به درخواست‌های کاربران نهایی از این API‌ها برای تعامل با یکدیگر استفاده می‌کنند. در کوبرنتیز، به منظور شبکه‌سازی برای ارتباط بین پادها و سرویس‌ها، به هر پاد یک آدرس IP منحصر به فرد اختصاص داده می‌شود که این امکان را فراهم می‌کند تا پادها بدون نیاز به NAT به صورت مستقیم با یکدیگر ارتباط برقرار کنند. با افزایش تعداد این میکروسرویس‌ها، مدیریت ارتباطات، امنیت و پایش این سرویس‌ها به چالشی بزرگ تبدیل می‌شود که می‌توان با Service Mesh آن را مدیریت کرد [۴۷]. سرویس مش یک لایه زیرساختی است که مدیریت ارتباط بین سرویس‌ها در معماری میکروسرویس‌ها را بر عهده دارد. این لایه قابلیت‌هایی مانند مشاهده‌پذیری، مدیریت ترافیک و امنیت را بدون تغییر کدهای برنامه اضافه می‌کند. Istio یک سیستم متن‌باز برای مدیریت اتصال، امنیت و مشاهده‌پذیری در معماری‌های میکروسرویس‌ها است که به عنوان سرویس مش شناخته می‌شود. این ابزار با افزودن یک لایه مستقل بین میکروسرویس‌ها و شبکه، توانایی‌هایی مانند مسیریابی هوشمند، ترافیک مدیریت شده، نظارت و امنیت را بهبود می‌بخشد. این سیستم از پروکسی‌های جانبه^{۱۹} برای کنترل ارتباطات بین میکروسرویس‌ها استفاده می‌کند که این پروکسی‌ها معمولاً از Envoy، یک پروکسی سریع و سبک، بهره می‌برند [۴۸]. از ویژگی‌های مهم Istio می‌توان به سه مورد زیر اشاره کرد:

- مدیریت ترافیک: به کاربران اجازه می‌دهد تا ترافیک بین سرویس‌ها را به صورت دقیق کنترل و مدیریت کنند. با استفاده از قابلیت‌های مسیریابی پیشرفته، کاربران می‌توانند قوانین پیچیده‌ای برای مسیریابی ترافیک تعریف کنند. این قوانین شامل

Tracking^{۱۷}
Packaging^{۱۸}
Sidecar proxies^{۱۹}

تقسیم بار^{۲۰}، مسیریابی مبتنی بر نسخه (برای پیاده‌سازی به روزرسانی‌های متوالی) و مدیریت ترافیک‌های خطای^{۲۱}

می‌شوند. این ویژگی‌ها به توسعه‌دهندگان کمک می‌کنند تا با اطمینان بیشتری به روزرسانی‌ها و تغییرات را در سیستم‌های

خود اعمال کنند.^[۴۸]

- امنیت: امکانات امنیتی جامعی برای ارتباطات سرویس به سرویس فراهم می‌کند. این امکانات شامل احراز هویت^{۲۲} و

مجوزدهی^{۲۳} مبتنی بر سیاست‌های امنیتی است. Istio با استفاده از MTLS ارتباطات بین سرویس‌ها را رمزنگاری

می‌کند و اطمینان حاصل می‌کند که فقط سرویس‌های معتبر می‌توانند با یکدیگر ارتباط برقرار کنند. این قابلیت‌ها به

افزایش امنیت سیستم‌های میکروسرویس کمک شایانی می‌کنند.

- مشاهده‌پذیری و نظارت: قابلیت‌های گستردۀ ای شامل جمع‌آوری و نمایش لاغ‌ها، متريک‌ها و تریس‌ها^{۲۴} برای

مشاهده‌پذیری و نظارت ارتباطات بین سرویس‌ها فراهم می‌کند. با استفاده از ابزارهای یکپارچه‌سازی‌شده مانند

Grafana و Prometheus کاربران می‌توانند به صورت جامع عملکرد و سلامت سیستم‌های خود را نظارت کنند.

همان‌طور که معماری این سیستم را در شکل ۶-۴ می‌بینید، Istio به دو بخش اصلی سطح داده^{۲۵} و سطح کنترل^{۲۶}

تقسیم می‌شود.

سطح داده

سطح داده مدیریت ارتباط بین سرویس‌ها را بر عهده دارد. در یک شبکه سنتی بدون سرویس مش، شبکه نمی‌تواند ترافیک را

بفهمد و بنابراین نمی‌تواند تصمیمات مبتنی بر نوع ترافیک یا منبع و مقصد آن بگیرد. با استفاده از این سیستم، هر ترافیک

شبکه‌ای که از سرویس‌ها خارج یا به آن‌ها وارد می‌شود توسط یک پروکسی رهگیری می‌شود. سطح داده شامل مجموعه‌ای از

پروکسی‌های هوشمند به نام Envoy است که به صورت Sidecar به هر سرویس در مش اضافه می‌شوند. این پروکسی‌ها وظیفه

میانجی‌گری و کنترل تمامی ارتباطات شبکه‌ای بین میکروسرویس‌ها را بر عهده دارند.^[۴۸]

Envoy پرآکسی، به عنوان یک پروکسی با عملکرد بالا، قابلیت‌های گستردۀ ای را برای مدیریت ترافیک میکروسرویس‌ها

در سرویس مش فراهم می‌کند. این قابلیت‌ها شامل کشف سرویس پویا است که به طور خودکار سرویس‌ها را شناسایی کرده و

Load balancing^{۲۰}

Fault injection^{۲۱}

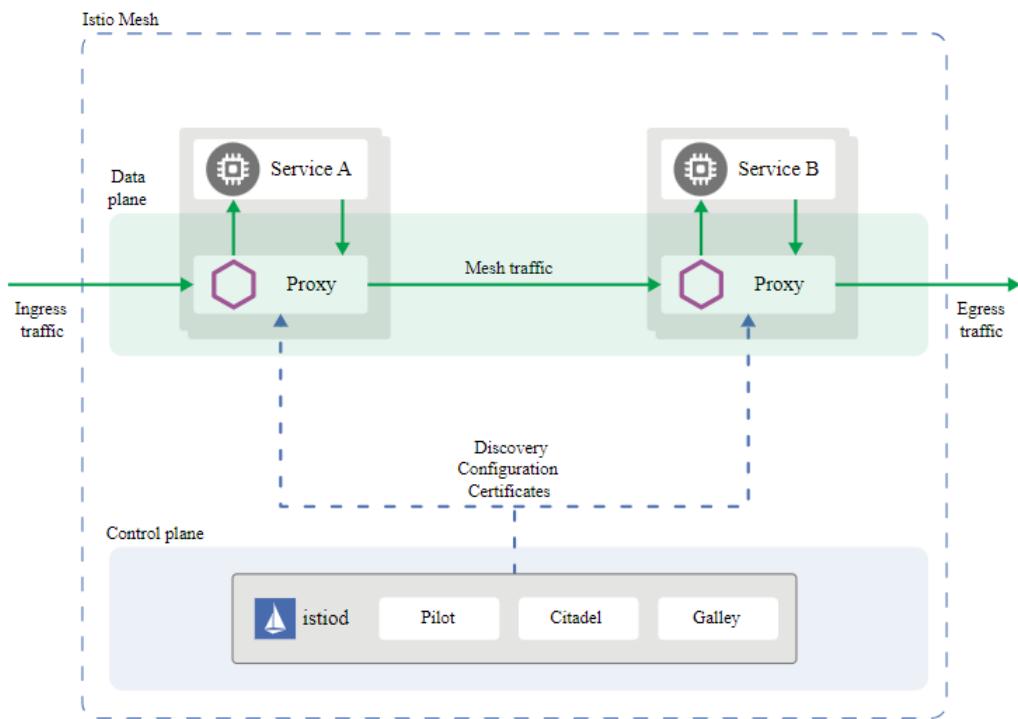
Authentication^{۲۲}

Authorization^{۲۳}

Trace^{۲۴}

Data plane^{۲۵}

Control plane^{۲۶}



شکل ۴-۶: معماری Istio

به تغییرات در توپولوژی پاسخ می‌دهد. تقسیم بار ترافیک را بهینه‌سازی می‌کند [۴۸]. این ابزار از پروتکل‌های 2/HTTP و gRPC برای بهبود کارایی پشتیبانی می‌کند و با استفاده از قطع کننده‌های مدار^{۲۷} از بارگذاری بیش از حد سرویس‌ها جلوگیری می‌کند. این پروکسی همچنین قابلیت بررسی سلامت سرویس‌ها، تزریق خطا برای شبیه‌سازی خرابی‌ها و جمع‌آوری تلمتری غنی را دارد.

سطح کنترل

سطح کنترل مسئول مدیریت و پیکربندی پروکسی‌های تشکیل‌دهنده سطح داده است. این سطح، پیکربندی مورد نظر شما را که از طریق سیاست‌ها و قوانین تعریف می‌شود، دریافت می‌کند و با استفاده از دید خود نسبت به سرویس‌ها در داخل مش، پروکسی‌ها را به صورت پویا برنامه‌ریزی می‌کند. هنگامی که قوانین یا محیط تغییر می‌کنند، سطح کنترل پروکسی‌ها را به روزرسانی می‌کند [۴۸]. این پیکربندی پویا به مدیریت ترافیک و سیاست‌ها به صورت بلاذرگ این امکان را می‌دهد تا سرویس مش به سرعت با تغییرات نیازهای برنامه یا زیرساخت‌ها سازگار شود.

اصلی‌ترین جزء صفحه کنترل Istiod نام دارد که مسئول کشف سرویس، مدیریت پیکربندی و مدیریت گواهی‌ها است. این ابزار، قواعد مسیریابی را به پیکربندی‌های Envoy تبدیل کرده و به سایدکارها ارسال می‌کند. Istiod به عنوان مرجع

صدور گواهی^{۲۸} عمل کرده و ارتباطات امن MTLS را در صفحه داده تضمین می‌کند. همچنین، با مدیریت هویت و اعتبارنامه‌ها، احراز هویت قوی و اجرای سیاست‌های امنیتی را فراهم می‌کند.

۳-۳-۴ مدیریت کاربران و چندمستاجری

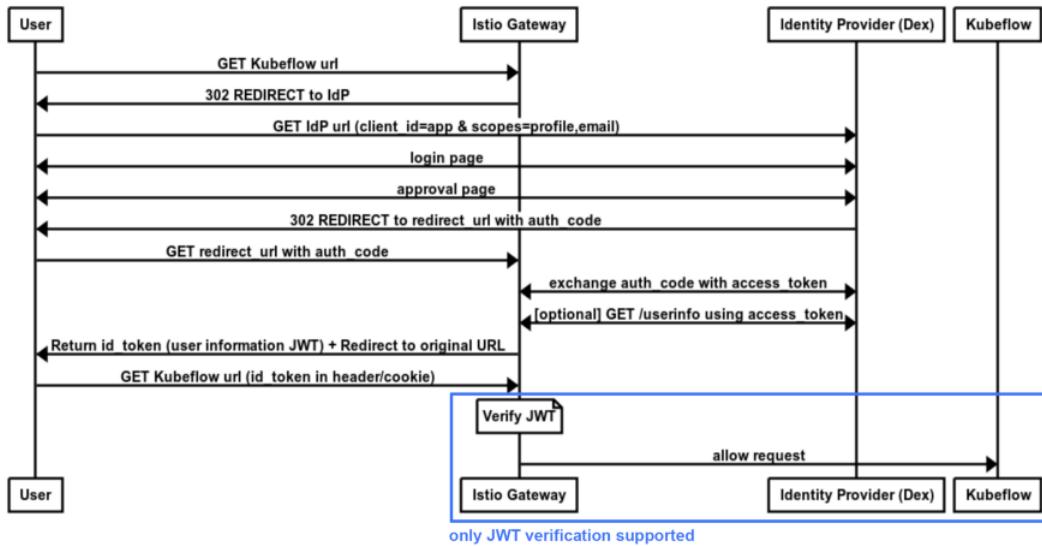
پشتیبانی از چندین کاربر و چندمستاجری^{۲۹} در پلتفرم ابری یکی از ویژگی‌های کلیدی است که به کاربران مختلف اجازه می‌دهد به صورت امن و مستقل بر روی همان پلتفرم کار کنند. این قابلیت با استفاده از مفاهیم ایزووله‌سازی، احراز هویت و مجوزدهی پیاده‌سازی شده است. در ادامه به جزئیات این فرآیندها می‌پردازیم.

احراز هویت

احراز هویت در پلتفرم از طریق ترکیبی از پروتکل OIDC^{۳۰} و Istio، به همراه ابزار Dex برای مدیریت هویت^{۳۱} و یکپارچگی با سیستم‌های احراز هویت خارجی، انجام می‌شود. پروتکل OIDC یک لایه احراز هویت بر روی OAuth 2.0 است که امکان تأیید هویت کاربران و دریافت اطلاعات پروفایل آن‌ها را فراهم می‌کند [۴۹]. Dex نیز یک سرویس منبع باز است که به عنوان یک ارائه‌دهنده هویت OIDC عمل می‌کند و می‌تواند با سیستم‌های هویتی مختلفی مانند LDAP و Active Directory یکپارچه شود.

شکل ۷-۴ روند احراز هویت یک کاربر با استفاده از پروتکل OIDC را نشان می‌دهد. کاربران ابتدا به رابط کاربری پلتفرم ابری وارد می‌شوند. در این مرحله، Dex به عنوان ارائه‌دهنده هویت، صفحه ورود را به کاربر نشان می‌دهد. سپس Dex اطلاعات کاربر را از سیستم‌های احراز هویت خارجی می‌گیرد و پس از تأیید هویت، یک توکن OIDC صادر می‌کند. پس از احراز هویت موفقیت‌آمیز، توکن OIDC به کاربر بازگردانده می‌شود و در هر درخواست HTTP به پلتفرم ابری، به عنوان هدر Authorization ارسال می‌شود [۴۹]. Istio به عنوان یک پروکسی معکوس عمل کرده و تمامی درخواست‌های ورودی را بررسی می‌کند. سرویس مش با استفاده از توکن OIDC ارسال شده توسط کاربر، هویت وی را تأیید می‌کند. پس از تأیید هویت توسط Istio، درخواست به سرور API پلتفرم ابری ارسال می‌شود. سرور API نیز هدرهای مربوط به اطلاعات هویتی را بررسی و پردازش می‌کند.

Certificate Authority ^{۲۸}
Multi-Tenancy ^{۲۹}
OpenID Connect ^{۳۰}
Identity Provider ^{۳۱}



شکل ۷-۴: روند احراز هویت

ایزوله‌سازی

ایزوله‌سازی در این پلتفرم با استفاده از کوبرنتیز namespace های کاربر مانند `app` می‌شود. هر namespace به یک یا چند کاربر اختصاص داده می‌شود و منابع هر کاربر مانند خط لوله‌ها و داده‌ها در namespace مخصوص به خود قرار می‌گیرند. این روش تضمین می‌کند که کاربران تنها به منابع خود دسترسی دارند و نمی‌توانند به منابع سایر کاربران دسترسی پیدا کنند.

مجوزدهی

مجوزدهی در پلتفرم ابری با استفاده از RBAC^{۳۲} کوبرنتیز انجام می‌شود. مجوزها به صورت نقش‌ها تعریف شده و از طریق RoleBinding به کاربران یا گروه‌های کاربران اختصاص داده می‌شوند [۴۴]. این روش به مدیران سیستم اجازه می‌دهد تا دسترسی‌های دقیقی برای کاربران تعیین کنند، مثلاً کاربری تنها بتواند خط لوله‌ها را مشاهده کند اما نتواند آن‌ها را اجرا کند.

^{۳۲} Role-based access control

۴-۳-۴ نظارت

نظارت در پلتفرم MLOps برای اطمینان از عملکرد بهینه و پایداری مدل‌های یادگیری ماشین از اهمیت ویژه‌ای برخوردار است. متريک‌ها در MLOps به سه دسته اصلی متريک‌های سистем، متريک‌های مدل و متريک‌های داده تقسیم می‌شوند.

متريک‌های سیستم

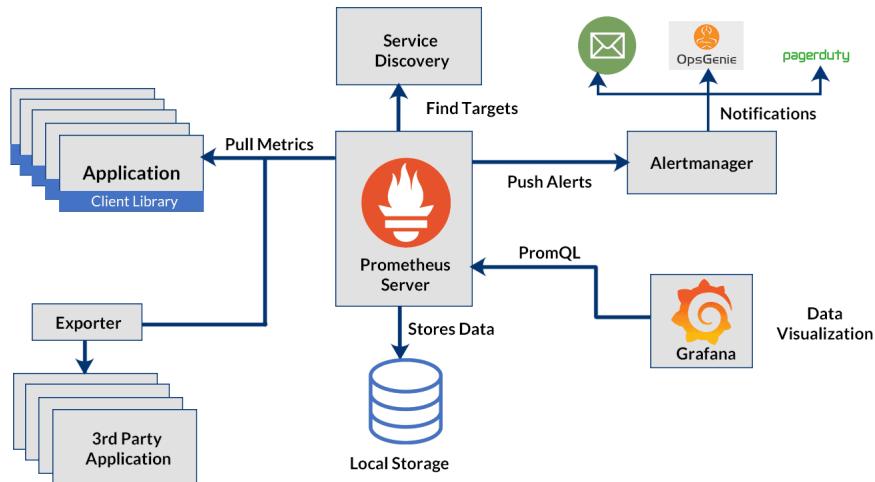
متريک‌های سیستم اطلاعاتی درباره مصرف منابع مانند CPU، حافظه، استفاده از ديسک و ترافيك شبکه را شامل می‌شوند. اين متريک‌ها به شناسايي مشكلات زيرساختی کمک می‌کنند که ممکن است بر عملکرد کلی سیستم تأثير بگذارند. مثلاً افزایش مصرف CPU یا حافظه می‌تواند نشان‌دهنده بار غيرعادی یا مشكلات در کد مدل باشد.

متريک‌های مدل

متريک‌های مدل برای ارزیابی عملکرد مدل‌های یادگیری ماشین استفاده می‌شوند و شامل پaramترهایی مانند مدت زمان پیش‌بینی، دقت، F1-score و نرخ خطأ هستند. مدت زمان پیش‌بینی نشان‌دهنده مدت زمان لازم برای انجام یک پیش‌بینی است و افزایش غيرعادی آن می‌تواند نشان‌دهنده مشكلات کارایی باشد. دقت مدل به ارزیابی کیفیت پیش‌بینی‌ها کمک می‌کند و کاهش در اين متريک‌ها ممکن است نشان‌دهنده نیاز به بازآموزی مدل^{۳۳} باشد. نرخ خطأ نیز به شناسايي مشكلات احتمالي در داده‌ها یا الگوريتم‌ها کمک می‌کند.

متريک‌های داده

متريک‌های داده شامل کیفیت داده‌ها، تغییرات در توزیع داده‌ها^{۳۴} و نرخ داده‌های مفقود می‌باشند. اين متريک‌ها برای تضمین کیفیت داده‌های ورودی و جلوگیری از بروز مشكلات ناشی از داده‌های نادرست یا ناکافی ضروری هستند. مثلاً تغییرات ناگهانی در توزیع داده‌ها می‌تواند نشان‌دهنده تغییرات در رفتار کاربران یا نقص در فرآيند جمع‌آوری داده‌ها باشد. نرخ داده‌های مفقود نیز به شناسايي مشكلات در جريان داده‌ها کمک می‌کند و از تأثيرگذاري منفي بر عملکرد مدل جلوگيری می‌کند. يكى از ابزارهای کلیدی برای جمع‌آوری اين متريک‌ها Prometheus است که برای جمع‌آوری، ذخیره‌سازی و مدیریت متريک‌ها به کار می‌رود. Prometheus از يك مدل داده مبتنی بر سري‌های زمانی و يك زبان پرس‌وجوی قدرتمند به نام PromQL استفاده می‌کند. متريک‌ها را از طریق scraping endpoint از Prometheus مشخص شده جمع‌آوری می‌کند.



شکل ۸-۴: معماری بخش نظارت

این endpointها می‌توانند توسط سرویس‌ها، اپلیکیشن‌ها و مدل‌های مختلف ارائه شوند که در آن‌ها متریک‌های مختلف به صورت فرمتهای مشخص مانند JSON قرار دارند. هر متریک می‌تواند دارای چندین برچسب باشد که به فیلتر کردن و گروه‌بندی داده‌ها کمک می‌کند. این برچسب‌ها می‌توانند شامل نام سرویس، نسخه مدل، نام مدل و دیگر اطلاعات مرتبط باشند [۵۰].

برای جمع‌آوری متریک‌ها، Prometheus به طور دوره‌ای به endpoint‌های مشخص شده مراجعه کرده و داده‌ها را دریافت می‌کند. این داده‌ها سپس در یک پایگاه داده زمان-سری ذخیره می‌شوند که امکان اجرای پرس‌وجوهای پیچیده و ایجاد هشدارهای مختلف را فراهم می‌کند. با تعریف آستانه‌های مختلف، Prometheus می‌تواند در صورت بروز شرایط بحرانی مانند افزایش غیرعادی زمان پیش‌بینی یا مصرف بیش از حد منابع، به تیم‌ها هشدار دهد [۵۰]. از Alertmanager نیز در کنار Prometheus برای مدیریت هشدارها استفاده می‌شود. هنگامی که یک قانون هشدار در Prometheus فعال می‌شود، هشدار به Alertmanager ارسال می‌شود. Alertmanager بر اساس پیکربندی‌های تعریف‌شده، هشدار را به گیرنده‌گان مناسب ارسال می‌کند. این گیرنده‌گان می‌توانند شامل ایمیل‌ها، سرویس‌های پیام‌رسانی (مانند Slack) و یا حتی runbooks برای اجرای خودکار اقدامات باشند. درنهایت، با استفاده از داشبوردهای تعاملی و قابل سفارشی‌سازی Grafana، تیم‌ها می‌توانند متریک‌های جمع‌آوری شده توسط Prometheus را به صورت بصری مشاهده و تحلیل کنند و با استفاده از فیلترهای مختلف، تحلیل‌های دقیقی از عملکرد مدل‌ها و سیستم‌ها ارائه دهند [۵۱] (شکل ۸-۴).

۵-۳-۴ استقرار مدل

استقرار مدل یکی از مراحل کلیدی در فرآیند استفاده از مدل‌های یادگیری ماشین است. در این مرحله، مدل‌های آموزش دیده به صورت سرویس‌های قابل دسترسی برای پیش‌بینی^{۳۵} به کار گرفته می‌شوند. استقرار مدل شامل مجموعه‌ای از فعالیت‌ها و فرآیندها می‌باشد که اطمینان حاصل می‌کند مدل‌ها به طور مؤثر و کارآمد در محیط تولید مورد استفاده قرار می‌گیرند.

اهمیت استقرار مدل در یادگیری ماشین به دلیل توانایی آن در انتقال مدل‌های آموزش دیده از محیط‌های توسعه به محیط‌های عملیاتی است که در آن‌ها می‌توانند ارزش واقعی ایجاد کنند. بدون استقرار مدل، تمامی تلاش‌ها و منابع صرف شده برای آموزش مدل‌ها بی‌فایده خواهند بود، زیرا مدل‌ها نمی‌توانند در دنیای واقعی مورد استفاده قرار گیرند. استقرار مدل امکان دسترسی سریع و کارآمد به پیش‌بینی‌های مدل را برای کاربران نهایی، برنامه‌ها و سیستم‌های دیگر فراهم می‌کند. این امر نه تنها به بهبود فرآیندهای تصمیم‌گیری و کارایی کسب‌وکار کمک می‌کند، بلکه از طریق نظارت مستمر و بهروزرسانی مدل‌ها، دقت و عملکرد آن‌ها نیز حفظ می‌شود. همچنین، استقرار مدل به سازمان‌ها اجازه می‌دهد تا مدل‌ها را به صورت مقیاس‌پذیر و با اطمینان از کارایی بالا به کار گیرند که این خود موجب بهبود تجربیات کاربری و افزایش بهره‌وری می‌شود [۵۲].

استقرار مدل‌های یادگیری ماشین به دو روش اصلی زیر انجام می‌شود:

۱. Embedded Model Serving: در استقرار جاسازی شده، مدل‌ها مستقیماً در برنامه اصلی تعییه می‌شوند. این روش باعث می‌شود که مدل‌ها بالاترین عملکرد را داشته باشند و زیرساخت ساده‌تری نیاز داشته باشند، زیرا همه‌چیز در یک واحد مجمع‌شده است. اما این روش نیازمند به روزرسانی مجدد برنامه در هر تغییر مدل است و مقیاس‌پذیری کمتری دارد، زیرا افزایش تعداد کاربران یا درخواست‌ها می‌تواند به کاهش عملکرد منجر شود. همچنین، باید تمام استراتژی‌های استقرار به صورت دستی پیاده‌سازی شوند که پیچیدگی بیشتری به همراه دارد [۵۲].

۲. Model as a Service (MaaS): در این نوع استقرار، مدل‌ها به عنوان سرویس‌های جداگانه ارائه می‌شوند که می‌توانند از هر برنامه‌ای در سازمان مورد استفاده قرار گیرند. این روش مزایای زیادی دارد از جمله ساده‌سازی ادغام مدل‌ها با سایر فناوری‌ها و فرآیندهای سازمانی، امکان استفاده مجدد از مدل‌ها در برنامه‌های مختلف، و پشتیبانی از دستگاه‌های کم قدرت که نمی‌توانند مدل‌های پیچیده را اجرا کنند. MaaS همچنین امکان استفاده از تکنیک‌های پیشرفت‌های مانند به روزرسانی مدل، تشخیص انحراف مدل و غیره را فراهم می‌کند [۵۲]. علاوه بر این، این روش اجازه می‌دهد که مدل‌ها و برنامه‌ها به طور مستقل مقیاس‌بندی شوند و می‌توانند بر روی سخت‌افزارهای مختلف مانند CPU و GPU اجرا شوند. اما معایبی از جمله کاهش عملکرد به دلیل وجود شبکه‌های اضافی و پیچیدگی‌های مدیریت هماهنگی زمانی بین

^{۳۵}Inference

سرویس مدل و برنامه اصلی نسبت به روش قبل دارد.

در استقرار مدل به عنوان سرویس نیز، دو رویکرد اصلی وجود دارد: مدل به عنوان کد^{۳۶} و مدل به عنوان داده^{۳۷}. در رویکرد اول، کد مدل به طور مستقیم در پیاده‌سازی سرویس استفاده می‌شود. به عبارت دیگر، کد مدل بخشی از کد اصلی سرویس است که در آن قرار دارد. این روش می‌تواند برای پروژه‌های کوچک یا محیط‌های توسعه مناسب باشد، زیرا ساده‌تر و مستقیم‌تر است. اما وقتی مدل‌ها به روزرسانی می‌شوند، نیاز به تغییر و استقرار مجدد سرویس وجود دارد که می‌تواند زمان برا و پیچیده باشد. در مقابل، رویکرد مدل به عنوان داده از یک پیاده‌سازی عمومی استفاده می‌کند که توسط یک مدل با فرمت استاندارد مانند PFA، ONNX یا فرمت اصلی TensorFlow استفاده می‌شود. این روش مزایای زیادی دارد، از جمله استانداردسازی تبادل مدل‌ها بین سرویس‌های مختلف که قابلیت جابه‌جایی بین سیستم‌ها را فراهم می‌کند [۵۲]. با استفاده از این رویکرد، مدل‌ها می‌توانند به صورت مستقل از سرویس‌ها به روزرسانی شوند و نیاز به استقرار مجدد سرویس‌ها نیست. بیشتر پیاده‌سازی‌های رایج در MaaS، ONNX Runtime، TFServing، Server TorchServe، و TorchServer از رویکرد مدل به عنوان داده استفاده می‌کنند و از فرمت‌های استاندارد بهره می‌برند.

الزامات و انتظارات از راه حل‌های استقرار مدل

استقرار مدل‌های یادگیری ماشین نیازمند درک و مدیریت جنبه‌های مختلفی است که شامل عملیات توسعه، تحلیل، آزمایش و مدیریت مدل‌ها می‌شود. این وظایف گسترده و پیچیده هستند و باید به صورت جامع مدیریت شوند تا مدل‌ها به درستی در محیط‌های عملیاتی اجرا شوند. در ادامه به بررسی مهم‌ترین الزامات و انتظارات از راه حل‌های استقرار مدل می‌پردازیم [۵۲].

- انعطاف‌پذیری فریمورک: یکی از مهم‌ترین الزامات، انعطاف‌پذیری فریمورک است. راه حل‌های استقرار مدل باید امکان

استفاده از مدل‌هایی که با فریمورک‌های مختلف مانند Scikit-learn، TensorFlow، PyTorch، و آموخته‌داند را فراهم کنند. این انعطاف‌پذیری به سازمان‌ها این امکان را می‌دهد که بدون نگرانی از تغییرات در فریمورک‌های زیرین، از همان رابط کاربری استفاده کنند و مدل‌ها را به سرعت به روزرسانی یا تغییر دهند.

- بهره‌گیری از بهینه‌سازهای سخت افزاری: قابلیت بهره‌گیری از بهینه‌سازهای سخت افزاری نظیر GPU‌ها و TPU‌ها یکی

دیگر از الزامات مهم است. بسیاری از مدل‌های یادگیری عمیق به دلیل پیچیدگی و عمق شبکه‌های عصبی نیاز به استفاده از این بهینه‌سازها دارند. استفاده از GPU‌ها و TPU‌ها می‌تواند زمان پیش‌بینی را کاهش داده و کارایی مدل را

بهبود بخشد.

^{۳۶} Model as Code
^{۳۷} Model as Data

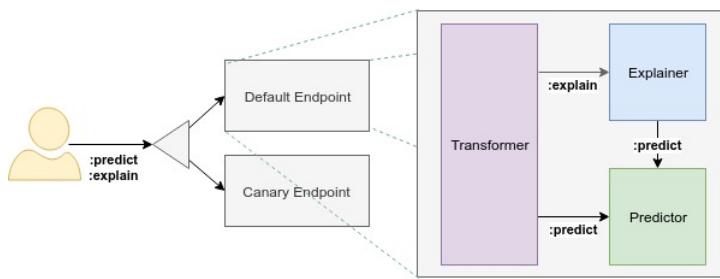
- مقياس‌پذیری: مقياس‌پذیری یکی دیگر از الزامات اساسی است. راه حل‌های استقرار باید امکان مقياس‌پذیری نمونه‌های استقرار را هم به صورت دستی و هم با استفاده از مقياس‌دهنده‌های خودکار فراهم کنند. این مقياس‌پذیری باید بدون

توجه به ساخت افزار زیرین انجام شود. همچنین، مقياس‌دهی هر یک از اجزای چرخه کاری استنتاج باید به صورت جداگانه انجام شود.

- پشتیبانی از درخواست‌های REST یا gRPC: راه حل‌های استقرار باید از درخواست‌های REST یا gRPC پشتیبانی کنند. این قابلیت‌ها امکان تعامل سریع و کارآمد با سرور مدل را فراهم می‌کنند و به انعطاف‌پذیری بیشتر در نحوه استفاده از مدل‌ها کمک می‌کنند.

به منظور استقرار مدل‌های یادگیری ماشین ابزارهای متan باز بسیاری وجود دارند که یکی از معروف‌ترین این ابزارها KServe می‌باشد. این ابزار یک راه حل پیشرفته برای استقرار و مدیریت مدل‌های یادگیری ماشین در محیط‌های تولیدی است. این ابزار به عنوان یک پروژه متan باز توسط Bloomberg، Google، Seldon، IBM توسعه یافته است و هدف آن پرکردن خلاهای موجود در ابزارهای مهم دیگر مانند TFServing و Seldon Core است. KServe با بهره‌گیری از اصول معماری Serverless و استفاده از زیرساخت‌های کوبرنتیو و Knative، پیچیدگی‌های مربوط به استقرار، مقياس‌پذیری، و مدیریت مدل‌های یادگیری ماشین را کاهش می‌دهد. علاوه بر این، این ابزار از مقياس‌پذیری دینامیک بر اساس رویدادها پشتیبانی می‌کند، که به معنای تخصیص منابع تنها در صورت نیاز است. ویژگی‌هایی مانند مقياس به صفر^{۳۸}، بهینه‌سازی استفاده از منابع و کاهش هزینه‌ها را تضمین می‌کنند. این امر همچنین به تیم‌ها اجازه می‌دهد تا مرکز بیشتری بر توسعه و بهبود مدل‌های خود داشته باشند، بدون نگرانی درباره پیچیدگی‌های زیرساخت [۵۳]. در کنار ویژگی‌های مهم یاد شده، KServe از فریم‌ورک‌های مختلف یادگیری ماشین مانند TensorFlow، XGBoost، Scikit-learn، PyTorch و Pystiebanی می‌کند. این قابلیت به کاربران اجازه می‌دهد تا مدل‌های خود را با استفاده از ابزارهای متan و مورد علاقه خود استقرار دهند.

از اجزای کلیدی در معماری KServe می‌توان به دو بخش سطح سرویس و سطح داده اشاره کرد. سطح سرویس در KServe شامل تمام پیکربندی‌ها و مدیریت‌های مربوط به سرورها و جریان‌های کاری است. این سطح با استفاده از Knative پیاده‌سازی شده و به طور خاص بر ارائه سرویس‌های Serverless مرکز دارد. Knative یک مجموعه از اجزا است که قابلیت‌های Serverless را برای کوبرنتیز فراهم می‌کند و شامل ویژگی‌هایی مانند مقياس‌پذیری خودکار، مدیریت ترافیک و استقرار آسان است. این امر به KServe اجازه می‌دهد تا به طور خودکار مقياس‌بندی کند، منابع را بر اساس نیاز تخصیص دهد و فرایندهای استقرار را ساده‌سازی کند. در مقابل، سطح داده در KServe شامل اجزایی است که وظایف مختلفی مانند



شکل ۹-۴: معماری سطح داده در K-Serve

مسیریابی درخواست‌ها، بررسی سلامت سرویس‌ها، تعادل بار، و احراز هویت و مجوزدهی را بر عهده دارند. اجزای کلیدی سطح داده شامل پیش‌بینی‌کننده، توضیح‌دهنده و مبدل می‌باشد. هسته اصلی سطح داده پیش‌بینی‌کننده است که مسئول اجرای مدل‌های یادگیری ماشین و ارائه پیش‌بینی‌ها می‌باشد. هر پیش‌بینی‌کننده شامل یک مدل و یک سرور مدل است که درخواست‌های ورودی را پردازش و پاسخ می‌دهد. توضیح‌دهنده یک جزء اختیاری است که توضیحات مدل را در کنار پیش‌بینی‌ها فراهم می‌کند. این توضیحات می‌توانند به کاربران کمک کنند تا درک بهتری از نتایج مدل داشته باشند و تصمیمات بهتری بگیرند. در آخر، مبدل نیز مسئول مراحل پیش‌پردازش و پس‌پردازش داده‌ها است. مبدل‌ها به کاربران اجازه می‌دهند تا داده‌ها را قبل از ارسال به مدل و پس از دریافت پیش‌بینی‌ها به شکلی خاص تبدیل کنند. این کار می‌تواند شامل پاکسازی داده‌ها، نرم‌افزاری، و تبدیل‌های دیگر باشد [۵۳] (شکل ۹-۴).

در کنار K-Serve به عنوان یک ابزار برای استقرار مدل به عنوان سرویس، ابزار معروف Seldon Core نیز وجود دارد. این پلتفرم بر پایه کوبرنتیز ساخته شده و قابلیت‌هایی مانند مسیریابی درخواست‌ها، مدیریت نسخه‌ها، نظارت بر عملکرد مدل و اندازه‌گیری متريک‌ها را فراهم می‌کند. با استفاده از Seldon Core، می‌توان مدل‌های مختلفی از جمله مدل‌های ساخته شده Scikit-learn و PyTorch، TensorFlow، با خصوصیت K-Serve را به راحتی مستقر و مدیریت کرد. در مقابل Kubeflow است و برای کاربرانی که از ابزارهای دیگر Kubeflow استفاده می‌کنند، یکپارچگی بهتری ارائه می‌دهد. Seldon Core مستقل از Kubeflow است و می‌تواند با هر پلتفرم کوبرنتیز کار کند. همچنین، Seldon Core از نظر پشتیبانی از الگوریتم‌های پیچیده‌تر مسیریابی درخواست و چارچوب‌های متنوع‌تر، انعطاف‌پذیری بیشتری دارد، در حالی که K-Serve بسادگی و یکپارچگی با اکوسیستم Kubeflow تمرکز دارد. در کنار این‌ها، K-Serve نیز با بهره‌بردن از قابلیت Serverless می‌تواند به مدیریت بهتر منابع کمک کند.علاوه بر این ابزار، ابزارهای TorchServe، TensorFlow Serving و Nvidia Triton به عنوان ابزارهای روشن استقرار جاسازی شده استفاده می‌شوند.

۶-۳-۴ جریان کاری یادگیری ماشین

جریان کاری در یادگیری ماشین به عنوان یکی از اجزای حیاتی در مدیریت و خودکارسازی جریان‌های کاری پیچیده در حوزه‌های مختلف از جمله یادگیری ماشین و مهندسی داده، نقش مهمی ایفا می‌کند. این سیستم‌ها از گراف‌های بدون حلقه جهت‌دار برای نمایش ترتیب اجرای وظایف استفاده می‌کنند. هر مرحله از این جریان کاری ممکن است شامل استخراج داده، آموزش مدل یا استنتاج باشد. این سیستم‌ها نه تنها ترتیب اجرای وظایف را مدیریت می‌کنند، بلکه وابستگی‌های متقابل بین وظایف را نیز مورد توجه قرار می‌دهند. همچنین این ابزارها به کاربران امکان می‌دهند تا جریان‌های کاری را به صورت خودکار و مقیاس‌پذیر اجرا کنند. این امر به ویژه در محیط‌های بزرگ با داده‌های کلان اهمیت دارد.

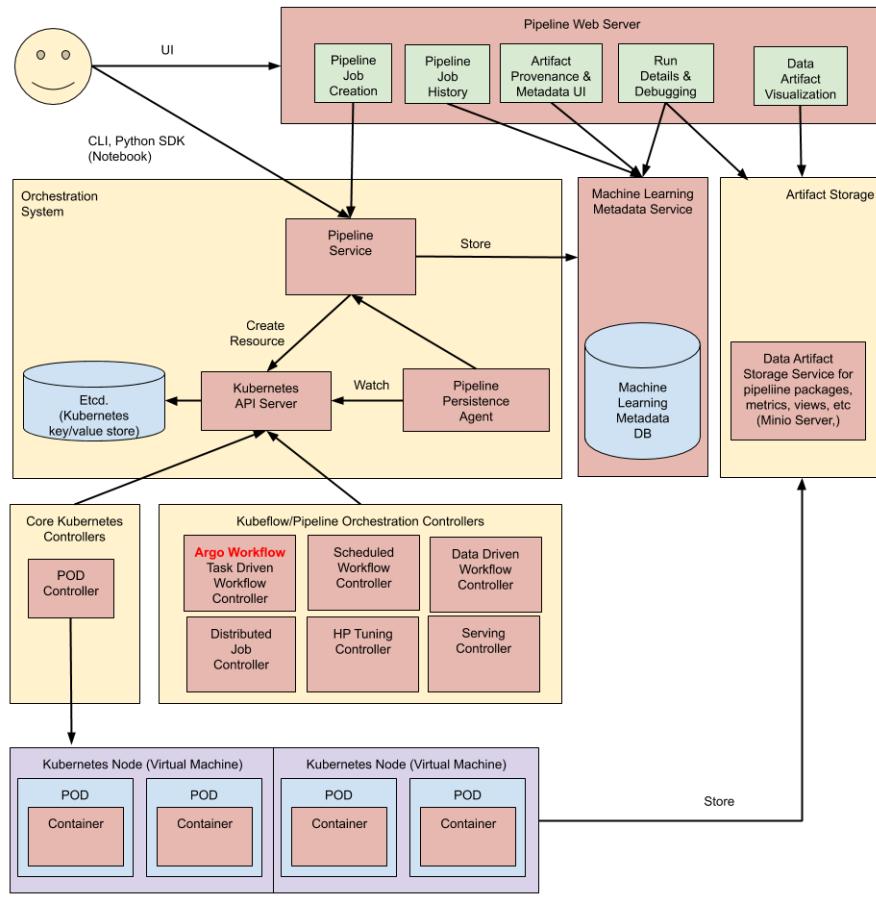
یکی از ابزارهای مهم و معروف برای پیاده‌سازی این قسمت، Kubeflow Pipelines می‌باشد. این ابزار یک پلتفرم قادر تمند برای مدیریت و ارکستراسیون جریان‌های کاری در زمینه یادگیری ماشین بر روی کوبرنتیز است. این پلتفرم از موتور جریان کاری Argo Workflows استفاده می‌کند که به طور طبیعی بر روی کوبرنتیز قرار دارد و برای اجرای خودکار این جریان‌ها به کار می‌رود. Kubeflow Pipelines با هدف ساده‌سازی فرآیند ساخت، مدیریت و اجرای خطوط لوله‌های یادگیری ماشین طراحی شده است. این پلتفرم از یک SDK پایتون و یک زبان خاص حوزه^{۳۹} برای تعریف جریان‌های کاری استفاده می‌کند که به کاربران این امکان را می‌دهد تا وظایف مختلف را به صورت برنامه‌نویسی شده مشخص کنند. از طریق این SDK، کاربران می‌توانند مراحل مختلف پردازش داده، آموزش مدل، و ارزیابی را به صورت کد پایتون ایجاد کرده و با هم مرتبط کنند. پس از تعریف و ایجاد مراحل خط لوله، کامپایلر DSL به کار می‌رود که کد پایتون را به فایل‌های YAML استاتیک تبدیل می‌کند. این فایل‌های YAML ساختار کامل خط لوله را به طور دقیق توصیف می‌کنند و به عنوان یک قالب استاتیک برای اجرای مراحل خط لوله در محیط کوبرنتیز استفاده می‌شوند. Pipeline Service در مرحله بعدی، این فایل‌های YAML را به CRD^{۴۰} کوبرنتیز تبدیل کرده و فرآیند اجرای خطوط لوله را مدیریت می‌کند. این فرآیند شامل ایجاد پادها و مدیریت مؤثر اجرای مراحل مختلف خطوط لوله است که توسط کنترلرهایی مانند Argo Workflow به صورت خودکار انجام می‌شود. این اجزا از معماری کوبرنتیز بهره می‌برند تا اجرای خطوط لوله را بهینه و مقیاس‌پذیر کنند که این امر از اهمیت بالایی برای ایجاد و مدیریت بارهای کاری یادگیری ماشین در محیط‌های توسعه و تولیدی برخوردار است (شکل ۱۰-۴).

بخش ذخیره‌سازی مؤلفه^{۴۱} نقش مهمی در Kubeflow Pipelines دارد. این بخش برای ذخیره‌سازی داده‌های تولید و مصرف شده توسط خطوط لوله استفاده می‌شود. داده‌های ذخیره شده به دو دسته اصلی تقسیم می‌شوند. دسته اول فراداده‌هایی

Domain Specific Language (DSL)^{۳۹}

Custom Resource Definition^{۴۰}

Artifact Storage^{۴۱}



شکل ۴: معماری Kubeflow Pipelines

شامل اطلاعاتی از جمله آزمایش‌ها، اجراهای و معیارهای متفرقه هستند. این اطلاعات در یک پایگاه داده MySQL ذخیره می‌شوند که امکان جستجو، مرتب‌سازی و فیلتر کردن آنها را فراهم می‌آورد. دسته دوم نیز مؤلفه‌هایی شامل بسته‌های خطوط لوله حاوی کدها، فایل‌های اجرایی، یا هر نوع فایل موردنیاز برای اجرای مراحل مختلف خطوط لوله و نمودارها است. این مؤلفه‌ها در MinIO ذخیره می‌شود. این سیستم ذخیره‌سازی به کاربران این امکان را می‌دهد تا داده‌های تولیدی و مصرفی خود را به طور کامل مدیریت کنند و به راحتی به آنها دسترسی داشته باشند که این امر برای تحلیل‌های بعدی، ارزیابی‌های کیفیت و اصلاحات مدل‌ها اهمیت زیادی دارد.

در کنار Apache Airflow، Kubeflow Pipelines یک ابزار متن‌باز و قابل تنظیم برای اجرای و مدیریت جریان‌های کاری است که ابتدا توسط Airbnb توسعه داده شد و سپس توسط Apache Software Foundation منتشر شد. Apache Airflow از یک معماری مبتنی بر گراف‌های بدون حلقه‌ی جهت‌دار استفاده می‌کند که به کاربران امکان می‌دهد تا جریان‌های کاری خود را همانند Kubeflow Pipelines به صورت گرافیکی تعریف کنند. Apache Airflow قابلیت‌هایی مانند زمان‌بندی مراحل کاری، اجرای موازی تسک‌ها، مدیریت وظایف متقاضی، مدیریت وابستگی‌ها و ارسال اعلان‌ها را فراهم می‌کند که بیشتر در زمینه‌های داده و

پردازش ETL استفاده می‌شود.

یکی از تفاوت‌های کلیدی بین این دو ابزار، سطح انتزاع و تخصص آن‌ها است. Airflow انعطاف‌پذیری بالایی دارد و می‌تواند برای طیف گسترده‌ای از گردش کارها به کار رود، اما نیازمند پیکربندی و کدنویسی بیشتری است. در مقابل، Kubeflow Pipelines با ارائه کامپوننت‌ها و توابع مخصوص به یادگیری ماشین، فرآیند ایجاد و مدیریت گردش کارهای یادگیری ماشین را ساده‌تر و مؤثرتر می‌سازد. همچنین، Kubeflow Pipelines به صورت بومی بر روی کوبرنتیز اجرا می‌شود که امکان مقیاس‌پذیری بالاتری را فراهم می‌آورد، درحالی‌که Airflow می‌تواند روی زیرساخت‌های مختلفی اجرا شود. علاوه بر این، Kubeflow Pipelines از سیستم‌های ذخیره‌سازی برای مؤلفه‌ها و فراداده‌ها استفاده می‌کند که برای مدیریت و تحلیل اطلاعات داده‌های تولید و مصرف شده توسط خطوط لوله موردنیاز در یادگیری ماشین بسیار حیاتی است. به طورکلی، Kubeflow Pipelines با امکانات و ویژگی‌های خود، بهترین گزینه برای توسعه، اجرا، و مدیریت جریان‌های کاری مرتبط با یادگیری ماشین است.

فصل ۱۰

پیاده‌سازی و نتایج

۱-۵ مقدمه

در این فصل به بررسی پیاده‌سازی پلتفرم و نتایج حاصل از آن می‌پردازیم. ابتدا به شرح سیستم مدیریت پلتفرم می‌پردازیم که نقش حیاتی در هماهنگی و یکپارچگی اجزای مختلف دارد و شامل استفاده از ابزارهای خودکارسازی و نظارت پیشرفته، مدیریت منابع سخت‌افزاری و بهروزرسانی مستمر می‌باشد. همچنین، فرآیند پیاده‌سازی خوش‌کوبرنیز برای پلتفرم MLOps و نصب مؤلفه‌های مختلف پروژه با استفاده از Helm شرح داده خواهد شد. در نهایت، نتایج حاصل از پیاده‌سازی پلتفرم و ارزیابی دو مسئله مختلف بررسی می‌شود: مسئله تشخیص ارقام و مسئله تحلیل احساسات در بازار سهام. این ارزیابی‌ها شامل بررسی عملکرد پلتفرم در شرایط مختلف و استفاده از ابزارهای متفاوت برای مدیریت و اجرای مدل‌های یادگیری ماشین به صورت خودکار و پیوسته است.

۲-۵ پیاده‌سازی پلتفرم

۱-۲-۵ سیستم مدیریت

در پلتفرم‌های بزرگ، نیاز به سیستمی برای مدیریت پلتفرم به وضوح احساس می‌شود. این سیستم باید قابلیت هماهنگی و یکپارچگی بین اجزای مختلف را داشته باشد تا اطمینان حاصل شود که همه بخش‌ها به درستی و بدون مشکل عمل می‌کنند. ویژگی‌های حیاتی این سیستم شامل استفاده از ابزارهای خودکارسازی و نظارت پیشرفته، مدیریت بهینه منابع سخت‌افزاری، پیاده‌سازی فرآیندهای مستمر بهبود و بهروزرسانی، مدیریت دسترسی‌ها و امنیت است. سیستم مدیریت شامل تمامی ابزارهای مدیریتی مانند مدیریت مخازن مؤلفه‌ها، کد و خط لوله CI/CD، مانیتورینگ کل سیستم و جمع‌آوری لاغ است. علاوه بر

جدول ۱-۵: مشخصات سخت افزاری ماشین های مدیریت

CPU	RAM	Storage	OS
4 Core	8 GB	512 GB	Ubuntu 18.04

این، استراتژی استقرار پروژه، اعمال مهاجرت‌ها^۱، پیکربندی پروژه‌ها، مدیریت سرورهای DNS و NTP^۲ و استفاده از ابزارهایی مانند Foreman برای نصب سیستم عامل به صورت PXE نیز توسط همین سیستم مدیریت می‌شود.

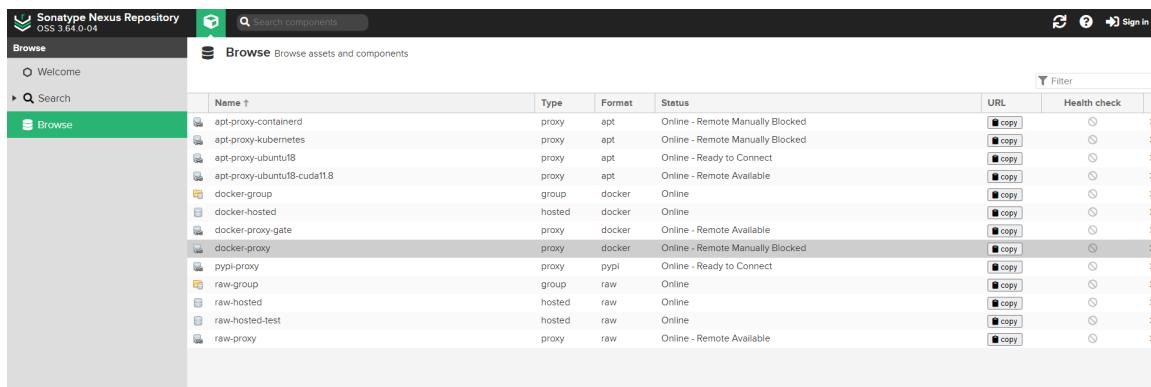
به منظور پیاده‌سازی این سیستم، ما دو ماشین مجزا برای مدیریت پلتفرم به منظور ایجاد قابلیت تحمل خطای^۳ و دسترسی پذیری بالا^۴ قرار می‌دهیم. از آنجایی که فرآیند و پروسه سنگینی روی این ماشین‌ها انجام نمی‌شود، مشخصات کمتری می‌تواند نسبت به ماشین‌های پروژه داشته باشد. مشخصات سخت افزاری هر کدام از این ماشین‌ها در جدول ۱-۵ قرار دارد. این ماشین‌ها به صورت ماشین مجازی با استفاده از OpenStack ساخته می‌شوند. ماشین‌های مدیریت باید برای نصب و راه‌اندازی ابزارهای مدیریت پلتفرم پیکربندی شوند و این کار با استفاده از ابزار Ansible انجام می‌گیرد. به این منظور Role‌های مشخصی برای هر بخش نوشته شده است تا بتوان بدون هیچ کار دستی و به صورت کاملاً خودکار سیستم‌ها را پیکربندی کرد. این Role‌ها با استفاده از قاعده نسخه‌گذاری Semantic نسخه‌گذاری شده و در مخزن raw موجود در Nexus که یک ابزار مدیریت مخازن مؤلفه می‌باشد نگهداری خواهد شد. در نهایت، برای پیکربندی سیستم، Role مورد نظر با نسخه مشخص از Nexus گرفته شده و با استفاده از Ansible ماشین‌ها پیکربندی می‌شوند. از آنجایی که این پیکربندی در محیط‌های مختلف مانند توسعه و عملیات می‌تواند متفاوت باشد، ما با استفاده از قابلیت Overriding در این ابزار مقادیر پیش‌فرض را برای هر محیط تغییر خواهیم داد. به همین منظور، اسکریپتی طراحی شده که در لینک گیت‌هاب^۵ قابل مشاهده است.

پروسه پیکربندی و نصب ابزار در ماشین‌های مدیریت به صورت زیر انجام می‌گردد:

۱. پیکربندی ماشین‌ها: این قسمت شامل نصب و پیکربندی ابزارهایی نظیر BIND برای سرور DNS، APT برای مدیریت ابزار در سیستم عامل Ubuntu، pip برای مدیریت کتابخانه‌های پایتون، chrony برای سرور NTP، LDAP برای مدیریت کاربران و ... می‌باشد.

۲. نصب و پیکربندی Docker: از آنجایی که مدیریت ابزارها به صورت کانتینر مناسب‌تر است، برای انجام مراحل بعدی نیاز به نصب Docker می‌باشد. پس از نصب، به منظور ذخیره‌سازی تمام مؤلفه‌ها مورد استفاده، به مخزن ساخته شده در Nexus مدیریت متصل خواهد شد.

Migrations^۱
Network Time Protocol^۲
Fault Tolerance^۳
High Availability^۴
<https://github.com/abolfazlyarian/mlops.git>^۵



شکل ۱-۵: مخازن مؤلفه در Nexus

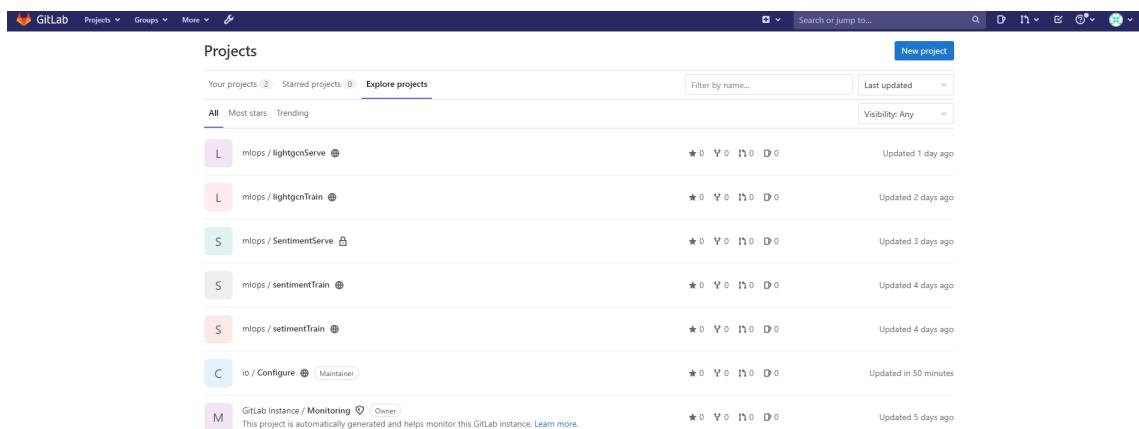
The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links like 'New Item', 'People', 'Build History', 'Project Relationship', 'Check File Fingerprint', 'Manage Jenkins', 'My Views', and 'Open Blue Ocean'. The main area is titled 'Build History' and displays a table of recent builds. The columns are S (Status), W (Workstation), Name, Last Success, Last Failure, Last Duration, and F (Filters). Four builds are listed: 'lightgcnServe', 'lightgcnTrain', 'sentimentServe', and 'sentimentTrain'. Each build entry includes a green circle icon, a cloud icon, and a date/time stamp. To the right of the table, there are three small icons: 'Icon legend', 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'. Below the table, there are two sections: 'Build Queue' (empty) and 'Build Executor Status' (listing 'Built-In Node' with 1 idle node).

شکل ۲-۵: خط لوله‌های CI/CD در Jenkins

۳. Nexus: از این ابزار به منظور مدیریت مخازن مؤلفه‌ها استفاده شده است. مخازن مورد استفاده ما APT، pip، Docker و raw می‌باشد (شکل ۱-۵).

۴. Jenkins: از این ابزار به منظور اجرا و مدیریت خط لوله‌های CI/CD پروژه‌ها و همچنین پیکربندی آنها توسط مدیران سیستم استفاده می‌شود (شکل ۲-۵).

۵. GitLab: به منظور مدیریت کد در پروژه‌ها و همچنین مدیریت Role‌های Ansible برای پیکربندی پروژه‌ها استفاده می‌شود (شکل ۳-۵).



شکل ۳-۵: مخازن کد در Gitlab

جدول ۲-۵: مشخصات سخت افزاری ماشین های خوشہ کوبرنتیز

CPU	RAM	Storage	OS
40 Core	128 GB	2 TB	Ubuntu 18.04

۲-۲-۵ خوشہ کوبرنتیز

در طراحی یک پلتفرم MLOps جامع و کارآمد که تمامی ابزارهای موردنیاز را در بر می‌گیرد، هدف اصلی ایجاد یک بستر یکپارچه، مقیاس‌پذیر و انعطاف‌پذیر برای مدیریت چرخه حیات مدل‌های یادگیری ماشین است. این پلتفرم شامل مجموعه‌ای از ابزارها و تکنولوژی‌های متن‌باز است که همگی روی خوشہ کوبرنتیز مستقر می‌شوند. استفاده از کوبرنتیز در پلتفرم‌های MLOps به دلیل قابلیت‌های منحصر‌به‌فرد آن در مدیریت خودکار، مقیاس‌پذیری و کارایی منابع است.

برای نصب خوشہ کوبرنتیز و انجام تست‌های اولیه پروژه MLOps، ۴ ماشین مجازی با مشخصات ذکر شده در جدول ۲-۵ استفاده شده است. در این پیاده‌سازی سه ماشین اول به عنوان گره اصلی و ماشین چهارم به عنوان گره کاری انتخاب شده است. همانند سیستم‌های مدیریت، ابتدا ماشین‌های مجازی که توسط OpenStack ساخته خواهند شد پیکربندی می‌شوند. پس از آن با استفاده از ابزار Kubeadm خوشہ کوبرنتیز پیاده‌سازی خواهد شد.

پیاده‌سازی خوشہ کوبرنتیز

برای نصب و پیاده‌سازی کوبرنتیز با استفاده از kubeadm، ابتدا پیش‌نیازهایی مانند Containerd و بسته‌های کوبرنتیز شامل kubelet، kubeadm و kubectl آماده می‌شوند. پس از پیکربندی سیستم و نصب Containerd، بسته‌های موردنیاز نصب شده و Swap غیرفعال می‌گردد، چرا که کوبرنتیز برای عملکرد بهینه نیاز به غیرفعال بودن Swap دارد. راهاندازی خوشہ با اجرای

دستور kubeadm init بر روی گره اصلی آغاز می‌گردد. این دستور تنظیمات اولیه را انجام داده و فایل پیکربندی برای kubectl ایجاد می‌کند که باید برای دسترسی به خوشة تنظیم گردد. پس از آن، نصب رابط شبکه کانتینر^۶ انجام می‌شود و در این مورد، ابزار Calico مورد استفاده قرار می‌گیرد. نصب Calico امکان برقراری ارتباط بین پادها در داخل خوشة را فراهم می‌سازد. در مرحله بعد، گره‌های کارگر به خوشه اضافه می‌شوند. دستورات لازم برای پیوستن گره‌های کارگر به خوشه که از دستور قبل به دست آمده‌اند، بر روی هر گره کارگر اجرا می‌گردد تا این گره‌ها به گره اصلی متصل شوند. در نهایت، وضعیت گره‌ها و صحبت پیوستن آن‌ها به خوشه بررسی می‌گردد و اطمینان حاصل می‌شود که همه گره‌ها به درستی اضافه شده و آماده به کار هستند. این روش نصب، راهی سریع و مؤثر برای راه‌اندازی خوشه کوبرنتیز فراهم می‌آورد که امکان اضافه کردن گره‌های جدید و مدیریت بهتر و مقیاس‌پذیری را به سهولت فراهم می‌سازد. تمامی این فرآیند با استفاده از Ansible Role‌های روی ماشین‌های پروژه انجام می‌گردد.

پیاده‌سازی مؤلفه‌های پروژه

تمامی مؤلفه‌های انتخاب شده در طراحی معماری فصل قبل با استفاده از Helm که وظیفه مدیریت مؤلفه‌ها را دارد انجام می‌گیرد. Helm یک ابزار قدرتمند برای مدیریت بسته‌های نرم‌افزاری کوبرنتیز است که فرآیند استقرار، بروزرسانی و مدیریت برنامه‌ها را ساده‌تر می‌کند. نصب یک چارت با استفاده از Helm این امکان را می‌دهد که برنامه‌ها و سرویس‌ها با کمترین تلاش ممکن و به صورت اتوماتیک در خوشه کوبرنتیز مستقر شوند.

یکی از مزایای استفاده از Helm، قابلیت تنظیم پارامترهای چارت‌ها در زمان نصب است. این قابلیت امکان سفارشی‌سازی چارت‌ها را براساس نیازهای خاص پروژه فراهم می‌کند. با استفاده از این ویژگی و قابلیت مشابه در Ansible، همانند قبل برای هر محیط مقادیر خاص آن محیط را پیکربندی می‌کنیم. علاوه بر این، Helm با ارائه قابلیت بازگشت‌پذیری، امکان بازگشت به نسخه‌های قبلی چارت‌ها را نیز فراهم می‌سازد که این ویژگی برای مدیریت نسخه‌ها و رفع مشکلات بسیار مفید است.

تمامی مؤلفه‌های اصلی نصب شده روی خوشه کوبرنتیز به منظور پیاده‌سازی پلتفرم MLOps در شکل ۴-۵ نشان داده شده است. نکته‌ای که راجع به این مؤلفه‌ها لازم به ذکر است، به منظور افزایش قابلیت اطمینان و تحمل خطأ، استفاده بیش از یک replica برای هر مؤلفه ضروری است. این امر باعث می‌شود تا در صورت خرابی یکی از نمونه‌ها، مؤلفه مربوطه به کار خود ادامه دهد و سرویس دهی دچار اختلال نشود. در شکل ۵-۵ نیز میزان مصرف منابع سخت‌افزاری سیستم زمانی که پلتفرم زیر بار نیست نشان داده شده است. همان‌طور که می‌بینید، میزان مصرف منابع سخت‌افزاری به صورت یکنواخت به خوبی

^۶ Container Network Interface (CNI)

NAMESPACE	NAME	READY	STATUS
auth	dex-6597dd7949-7lln6	1/1	Running
cert-manager	cert-manager-5dfdbefdf6b-4qw26	1/1	Running
cert-manager	cert-manager-cainjector-8944dc75f-brbgz	1/1	Running
cert-manager	cert-manager-webhook-6985f5b7d-6b95r	1/1	Running
granite	granite-alertmanager-0	2/2	Running
granite	granite-elastic-monitoring-io-10-657f5f657d-nm8kf	1/1	Running
granite	granite-elastic-monitoring-io-11-9cd6d857b-gqmkk	1/1	Running
granite	granite-elastic-monitoring-io-9-77bb785878-245m9	1/1	Running
granite	granite-filebeat-9b5x6	2/2	Running
granite	granite-filebeat-gxvf6	2/2	Running
granite	granite-filebeat-hsx6	2/2	Running
granite	granite-filebeat-lh7mc	2/2	Running
granite	granite-grafana-add-objects-job-fbd2c	0/1	Succeeded
granite	granite-grafana-deploy-568fc5856-n5tzh	3/3	Running
granite	granite-kube-state-metrics-deploy-7bdbf4974f-xmdtr	1/1	Running
granite	granite-minio-0	1/1	Running
granite	granite-minio-1	1/1	Running
granite	granite-minio-2	1/1	Running
granite	granite-minio-3	1/1	Running
granite	granite-node-exporter-5wntx	1/1	Running
granite	granite-node-exporter-6cm99	1/1	Running
granite	granite-node-exporter-qmbz	1/1	Running
granite	granite-node-exporter-tdqcm	1/1	Running
granite	granite-ohab-6c7bf4f45d-wxsx6	1/1	Running
granite	granite-openeps-localpv-7776f556bb-mkpqx	1/1	Running
granite	granite-postgresql-0	2/2	Running
granite	granite-postgresql-1	2/2	Running
granite	granite-postgresql-2	2/2	Running
granite	granite-postgresql-operator-c9b64fbb-b4rbx	1/1	Running
granite	granite-postgresql-0	1/1	Running
granite	granite-postgresql-1	1/1	Running
granite	granite-postgresql-2	1/1	Running
granite	granite-prometheus-0	2/2	Running
granite	granite-prometheus-1	2/2	Running
granite	granite-redis-595c897554-8955x	2/2	Running
granite	sms-gateway-deploy-56575fb9d-n9lzt	1/1	Running
istio-system	authservice-0	1/1	Running
istio-system	cluster-local-gateway-5c44b4f6b7-z9v8k	1/1	Running
istio-system	istio-ingressgateway-7b5c8644b4-9gmdp	1/1	Running
istio-system	istiod-77c5984b95-zw9bn	1/1	Running
knative-serving	activator-5d9f58896d-96t92	1/1	Running
knative-serving	autoscaler-594f86cb56-4jzmn	1/1	Running
knative-serving	controller-548bd754-l8b9x	1/1	Running
kserve	kserve-controller-manager-0	2/2	Running
kube-system	calico-kube-controllers-577f77cb5c-gfrgt	2/2	Running
kube-system	calico-node-4tf5z	1/1	Running
kube-system	calico-node-c79k8	1/1	Running
kube-system	calico-node-lwj6w	1/1	Running
kube-system	calico-node-sp2jx	1/1	Running
kube-system	calico-typha-7487b564bd-5wf46	1/1	Running
kube-system	calico-typha-7487b564bd-grt48	1/1	Running
kube-system	etcd-io-10	1/1	Running
kube-system	etcd-io-11	1/1	Running
kube-system	etcd-io-9	1/1	Running
kube-system	granite-coredns-57755dcc7d-rccrl	1/1	Running
kube-system	granite-coredns-57755dcc7d-rp2kk	1/1	Running
kube-system	granite-metrics-server-77988b6d66-8gfn	1/1	Running
kube-system	kube-apiserver-io-10	1/1	Running
kube-system	kube-apiserver-io-11	1/1	Running
kube-system	kube-apiserver-io-9	1/1	Running
kube-system	kube-controller-manager-io-10	1/1	Running
kube-system	kube-controller-manager-io-11	1/1	Running
kube-system	kube-controller-manager-io-9	1/1	Running
kube-system	kube-proxy-56fjs	1/1	Running
kube-system	kube-proxy-gbxjf	1/1	Running
kube-system	kube-proxy-lzfhf	1/1	Running
kube-system	kube-proxy-xzklw	1/1	Running
kube-system	kube-scheduler-io-10	1/1	Running
kube-system	kube-scheduler-io-11	1/1	Running
kube-system	kube-scheduler-io-9	1/1	Running
kubeflow-user-example-com	digits-recognizer-0	2/2	Running
kubeflow-user-example-com	ml-pipeline-ui-artifact-6bcd5db8c4-s8pn1	2/2	Running
kubeflow-user-example-com	ml-pipeline-visualizationserver-58f98f845-jz9jv	2/2	Running
kubeflow-user-example-com	stock-lstm-predictor-default-00001-deployment-85cd5d54f8-jl7th	2/2	Running
kubeflow	admission-webhook-deployment-67b69f9f6f-65cpk	1/1	Running
kubeflow	cache-server-8698946867-tmwzk	2/2	Running
kubeflow	centraldashboard-659996b95f-5nljq	2/2	Running
kubeflow	jupyter-web-app-deployment-7b78b6b8b9-b8clw	1/1	Running
kubeflow	katib-controller-789b7b87d-c19wm	1/1	Running
kubeflow	katib-db-manager-6ff7d6d45d-sqjb5	1/1	Running
kubeflow	katib-mysql-5488655f77-4n836	1/1	Running
kubeflow	katib-ui-5f675fdc8-sh2f4	1/1	Running
kubeflow	kserve-models-web-app-77c898cc4c-kxwll	2/2	Running
kubeflow	kubeflow-pipelines-profile-controller-6db748cf8b-k4wjh	1/1	Running
kubeflow	metacontroller-0	1/1	Running
kubeflow	metadata-envoy-deployment-7df947d668-nhrjr	1/1	Running
kubeflow	metadata-gRPC-deployment-8bbbbbfb8c-zb2kr	2/2	Running
kubeflow	metadata-writer-6fdccfb6-gf58d	2/2	Running
kubeflow	minio-75d7f88454-sj4lt	2/2	Running
kubeflow	ml-pipeline-7cf9c4f6d5-9x6kn	2/2	Running
kubeflow	ml-pipeline-persistenceagent-cff6bd654-6crnk	2/2	Running
kubeflow	ml-pipeline-scheduledworkflow-848cb4c7fc-6vvvcz	2/2	Running
kubeflow	ml-pipeline-ui-6fbb699cb6-gl77c	2/2	Running
kubeflow	ml-pipeline-viewer-crd-578699d8fb-wt9pr	2/2	Running
kubeflow	ml-pipeline-visualizationserver-6696c655c9-n47rv	2/2	Running
kubeflow	mysql-5f554874bf-x16gj	2/2	Running
kubeflow	notebook-controller-deployment-66b9f45ffb-hfbzb	2/2	Running
kubeflow	profiles-deployment-bdfb879fc-gsrsv	3/3	Running
kubeflow	tensorboard-controller-controller-manager-74dd6467c8-mrg95	3/3	Running
kubeflow	tensorboards-web-app-deployment-7c6fc9bc4c-vq7hw	1/1	Running
kubeflow	training-operator-65c69fc75-6w4qn	1/1	Running
kubeflow	volumes-web-app-deployment-7474c6d7-r4h9b	1/1	Running
kubeflow	workflow-controller-58796bfd74-rkmdd	2/2	Running

شکل ۴-۵: وضعیت مؤلفه‌ها در خوشه کوبرنیتیز

NAME	CPU(cores)	CPU%	MEMORY(bytes)	MEMORY%
io-10	875m	2%	15024Mi	15%
io-11	918m	2%	14920Mi	15%
io-12	415m	1%	13491Mi	14%
io-9	1087m	2%	17167Mi	17%

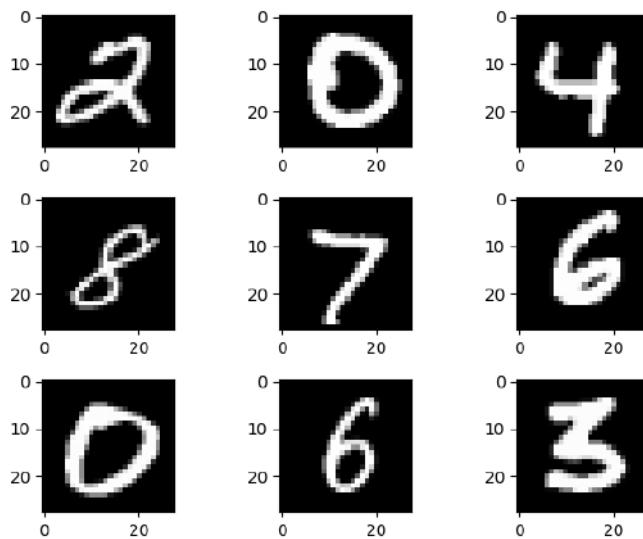
شکل ۵: میزان مصرف منابع سخت‌افزاری پلتفرم بدون بار

شکل ۶: داشبورد پلتفرم MLOps

توزیع شده است که منجر به استفاده بهینه از منابع سخت‌افزاری می‌گردد. در شکل ۶ نیز داشبورد پلتفرم نشان داده شده است. در این داشبورد ما سرویس‌های Notebooks برای مدیریت جوپیتر نوت‌بوک‌های کاربر، TensorBoard به‌منظور مشاهده و تحلیل نتایج مدل‌های یادگیری عمیق، Models برای مدیریت مدل‌های استقرار یافته و Runs به‌منظور مدیریت چرخه‌های کاری یادگیری ماشین لحاظ شده است.

۳-۵ حل مسئله و نتایج

پس از پیاده‌سازی پلتفرم MLOps نیاز به ارزیابی آن داریم. هدف از این‌کار، بررسی قابلیت‌ها و عملکرد این پلتفرم در مدیریت و اجرای مدل‌های یادگیری ماشین به صورت خودکار و پیوسته است. به‌منظور ارزیابی و تست پلتفرم، دو مسئله مختلف با دو رویکرد متفاوت مورد بررسی قرار می‌گیرند. در مسئله اول، از قابلیت Notebooks در این پلتفرم استفاده شده است و در مسئله دیگر که به صورت کدهای پایتون نوشته شده است با بهره‌گیری از CI/CD، روی پلتفرم پیاده‌سازی و اجرا می‌شوند. این رویکرد، به ما امکان می‌دهد تا عملکرد پلتفرم را در شرایط مختلف و با استفاده از ابزارهای متفاوت بررسی و ارزیابی کنیم.



شکل ۷-۵: نمونه داده مجموعه دادگان MNIST

۱-۳-۵ مسئله تشخیص ارقام

مسئله تشخیص ارقام دستنویس یکی از مسائل پایه‌ای در حوزه یادگیری عمیق و بینایی ماشین است. این مسئله با استفاده از مجموعه دادگان MNIST شامل ۶۰۰۰۰ تصویر آموزشی و ۱۰۰۰۰ تصویر آزمایشی از ارقام ۰ تا ۹ حل شده‌است. هر تصویر دارای ابعاد 28×28 پیکسل و سیاه و سفید می‌باشد (شکل ۷-۵). برای تشخیص این ارقام، معمولاً از مدل‌های شبکه عصبی پیچشی^۷ استفاده می‌شود که با بهره‌گیری از لایه‌های کانولوشن و پولینگ، ویژگی‌های مکانی تصاویر را استخراج می‌کنند. این شبکه‌ها با استفاده از فیلترهای قابل یادگیری، ویژگی‌های محلی و پیچیده‌ای مانند لبه‌ها، گوش‌ها و الگوهای خاص را شناسایی می‌کنند. پس از چندین لایه کانولوشن و پولینگ، شبکه به یک لایه کاملاً متصل^۸ یا چند لایه می‌رسد که وظیفه دسته‌بندی نهایی را بر عهده دارند.

چرخه کاری آموزش

رویکرد حل مسئله ما در این بخش، استفاده از قابلیت Notebooks در پلتفرم می‌باشد. لذا از خط‌لوله‌های CI/CD برای این منظور استفاده نشده‌است. در ابتدا مجموعه دادگان MNIST در یک باکت مجزا در MinIO ذخیره‌سازی می‌شود. با استفاده از کتابخانه‌های پایتونی به محل ذخیره‌سازی متصل شده و داده‌ها گرفته و در کانتینری که به همین منظور به کاربر داده می‌شود ذخیره‌سازی می‌شوند. پس از مرحله پیش‌پردازش، داده‌ها برای آموزش مدل کانولوشنی مورد استفاده قرار می‌گیرند. پس از

Convolutional neural network (CNN)^V

Fully Connected^A

جدول ۳-۵: زمان اجرا چرخه یادگیری ماشین در مسئله تشخیص ارقام

خط لوله	زمان (ثانیه)
چرخه کاری آموزش	98
چرخه کاری استقرار	69

آموزش، مدل نهایی را در MinIO ذخیره‌سازی می‌کنیم.

چرخه کاری استقرار

همانند گذشته، از قابلیت نوتبوک در پلتفرم برای استقرار مدل بهره برده‌ایم. در اینجا با استفاده از استقراردهنده‌های مخصوص K-Serve برای استقرار مدل بهره برده‌ایم. با توجه به اینکه چارچوب TensorFlow برای مدل استفاده شده است، از استقراردهنده TensorFlow Serving بهره گرفته می‌شود. به منظور استقرار مدل، تنها کافی است که آدرس ذخیره‌سازی مدل که همان آدرس MinIO آن است را به همراه نوع استقراردهنده مورد نظر مشخص کنید.

نتایج

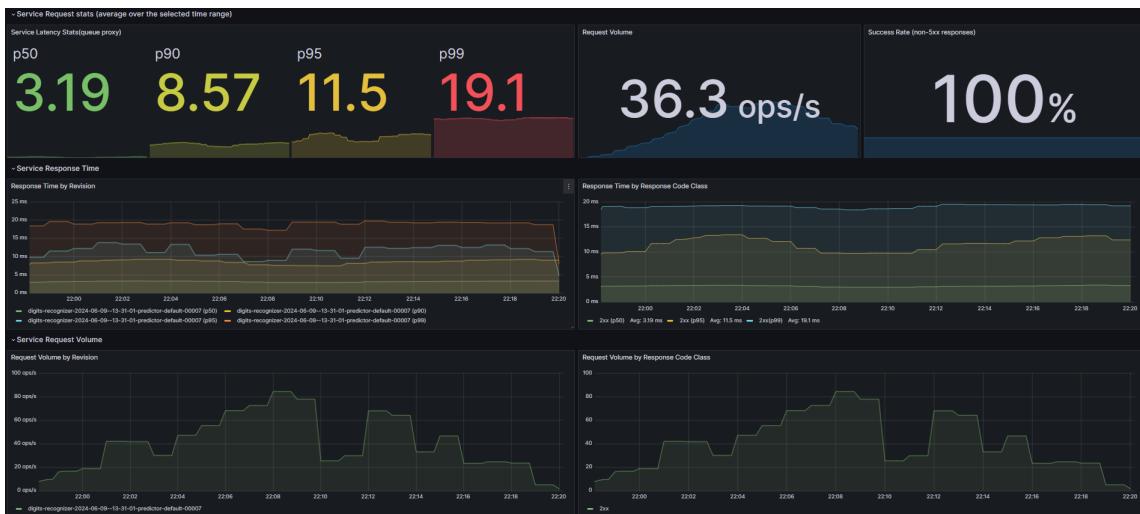
از آنجا که دقت و کارایی مدل هدف پیاده‌سازی مسئله نبوده و صرفاً هدف ما تست عملکردی پلتفرم می‌باشد، ملاک ارزیابی ما زمان اجرای خط لوله‌ها و چرخه یادگیری ماشین، مدت زمان پاسخ و میزان مصرف منابع می‌باشد. در جدول ۳-۵ زمان اجرای هر بخش را برای اجرای یک تغییر نشان داده شده است. کل مدت زمان آموزش تا استقرار مدل مجموعاً ۱۶۷ ثانیه می‌باشد. توجه داشته باشید که این زمان بدون لحاظ زمان کش می‌باشد. در صورتی که تغییر در یک جزء صورت می‌گیرد تنها آن جزء اجرا شده و بقیه اجزاء از حافظه نهان برای اجرا آن استفاده می‌کنند.

به منظور تست مدل استقرار یافته، به صورت همزمان به آن درخواست داده و زمان پاسخ را گزارش کرده‌ایم. نتایج این تست در جدول ۴-۵ گزارش شده است. برای یک درخواست ارسال شده به مدل، مدت زمان استنتاج مدل تقریباً بین ۳ تا ۸ میلی ثانیه و بقیه زمان ارسال و دریافت درخواست از طریق Istio gateway می‌باشد (شکل ۸-۵).

نتایج در تعداد درخواست‌های بالا به دلیل استفاده از تکنیک مقیاس‌پذیری خودکار در پلتفرم با استفاده از ابزارهای Knative و Istio مناسب است ولی در تعداد درخواست‌های پایین مناسب نمی‌باشد. به منظور رفع این مشکل از سرویس LoadBalancer در خوشکوبرنریز استفاده شده است که نتایج آن را در شکل ۹-۵ مشاهده می‌کنید. در این روش به دلیل استفاده نکردن از تکنیک مقیاس‌پذیری خودکار، زمان درخواست‌ها در سناریوهایی که تعداد درخواست‌های همزمان زیاد می‌باشد، مناسب نمی‌باشد.

جدول ۴-۵: تست استنتاج مدل مسئله تشخیص ارقام

تعداد	زمان پاسخ (ثانیه)
1	4.03
4	4.04
10	4.07
16	4.1
32	4.18
64	4.34
100	4.43
128	4.66
256	5.12
512	6.36

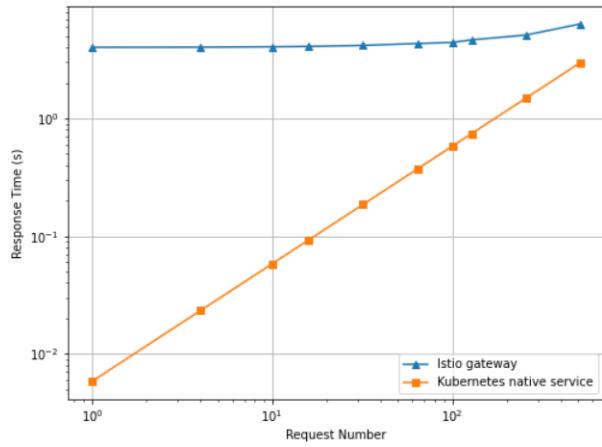


شکل ۸: گراف استنتاج مدل در مسئله تشخیص ارقام (داشبورد Grafana)

به منظور تست دسترسی نوتبوک به پردازنده گرافیکی به صورت تستی یکبار چرخه کاری آموزش با فعال کردن قابلیت آموزش بر روی پردازنده گرافیکی انجام شده است. نتایج استفاده از پردازنده گرافیکی نیز در شکل ۱۰-۵ نشان داده شده است.

۲-۳-۵ مسئله تحلیل احساسات در بازار سهام

مسئله اصلی این پروژه تحلیل احساسات در بازار سهام است. هدف این است که با استفاده از نظرات و اخبار اقتصادی، احساسات مثبت و منفی موجود در بازار شناسایی شود و تأثیر آن بر رفتار بازار تحلیل گردد. داده‌های دادگان این مسئله به تجربیات احساس بازار سهام مربوط می‌شود که از حساب‌های مختلف توثیق شده اند. شامل ۵۷۹۱ توثیق مرتبط با اخبار اقتصادی پیرامون سهام، با دسته‌بندی احساس مثبت و منفی جمع آوری شده‌اند. این مجموعه داده به دو بخش مثبت و منفی تقسیم



شکل ۵-۵: زمان استنجاج مدل در مسئله تشخیص ارقام

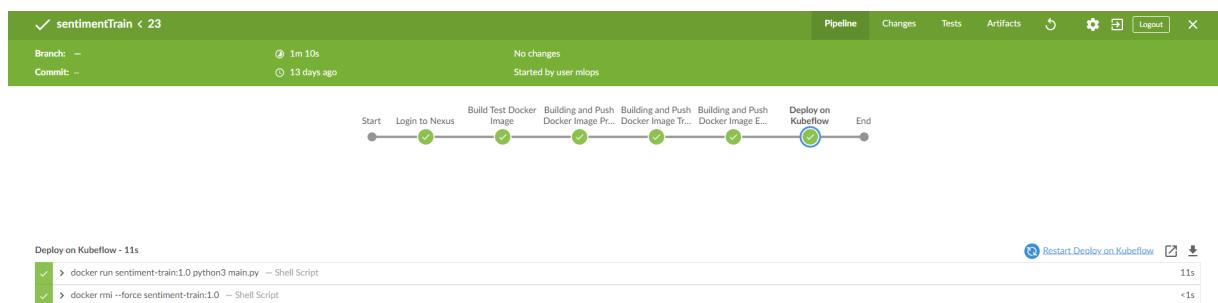


شکل ۱۰-۵: گراف نظارت بر پردازنده گرافیکی در مسئله تشخیص ارقام (داشبورد Grafana)

می‌شوند که تعداد موارد منفی ۲۱۰۶ و تعداد موارد مثبت ۳۶۸۵ است. در شکل ۱۱-۵ ده سطر اول از این داده‌ها به عنوان نمونه نشان داده شده است. راه حل ارائه شده برای حل این مسئله استفاده از شبکه‌های LSTM می‌باشد. به منظور کنترل و مدیریت این مسئله از دو خط لوله مجزا برای آموزش و استقرار استفاده شده است.

index	text	sentiment
1	Kickers on my watchlist XIDE TIT SQQ PNK CPW BPZ AJ trade method 1 or method 2, see prev posts	1
2	user: AAP MOVIE. 55% return for the FEA/GEED indicator just 15 trades for the year. AWESOME.	1
3	user I'd be afraid to short AMZN - they are looking like a near-monopoly in eBooks and infrastructure-as-a-service	1
4	MNTA Over 12.00	1
5	OI Over 21.37	1
6	PGNX Over 3.04	1
7	AAP - user if so then the current downtrend will break. Otherwise just a short-term correction in med-term downtrend.	-1
8	Monday's relative weakness. NYX WIN TIE TAP ICE INT BMC AON C CHK BIIB	-1
9	GOOG - over trend line channel test & volume support.	1
10	AAP will watch tomorrow for ONG entry.	1

شکل ۱۱-۵: مجموعه داده احساسات در بازار سهام



شکل ۵: خط لوله آموزش مسئله تحلیل احساسات در بازار سهام

خط لوله آموزش

در این خط لوله از قابلیت کانتینر برای استقرار آسان استفاده شده است و شامل آزمون هایی برای اطمینان از عملکرد و دقت راه حل های پیاده سازی شده است که هر کانتینر با ساخت و اجرای فایل و تصویر داکر مربوطه اجرا خواهد شد. همچنین، برای ساخت خط لوله در بستر پلتفرم لازم است که در ابتدا بخش های مختلف پروژه را به مولفه های مجزا تقسیم کرد، به طوری که هر مولفه دارای یک فایل داکر بوده و به صورت مجزا به آن نگاه خواهد شد. در اینجا ما با سه مولفه سروکار خواهیم داشت که مراحل پیش پردازش، آموزش و آزمایش چرخه‌ی یادگیری ماشین ما خواهند بود. همچنین برای ساخت خط لوله نیاز است تا اسکریپت جداگانه‌ای مربوط با کدهای این بخش که با کتابخانه kfp است نوشته شود، تا بتوان با استفاده از تصاویر داکری ساخته شده در هر مولفه، پایپلاین مورد نظر را روی بستر پلتفرم MLOps ساخت. در شکل ۱۲-۵ یکی از اجراهای خط لوله CI/CD برای مرحله آموزش را مشاهده می‌کنید. همان‌طور که می‌بینید این خط لوله شامل چندین بخش از جمله پیش‌پردازش، آموزش، ارزیابی و پیاده‌سازی روی بستر پلتفرم MLOps با استفاده از Kubeflow Pipeline می‌باشد.

پیش‌پردازش شامل چندین مرحله کلیدی است که داده‌ها را برای مدل‌سازی آماده می‌کند. این مراحل شامل تغییر برچسب‌ها به مقادیر دسته‌ای، تمیز کردن متن از کاراکترها و کلمات بی‌معنی، تقسیم داده‌ها به دو بخش آموزش و ارزیابی، و تبدیل متن به دنباله‌هایی از کلمات است. همچنین، واژگان با استفاده از تعبیه‌گرهای پیش‌آموزش داده شده GloVe به بردارهایی تبدیل می‌شوند که حاوی اطلاعات معنایی و ارتباطی کلمات هستند. فایل‌های تعبیه‌گر GloVe از MinIO دانلود شده و بردارهای تعبیه برای واژگان موجود در داده‌های آموزش ساخته می‌شود. این بردارها سپس برای ایجاد یک ماتریس تعبیه جهت استفاده در شبکه عصبی به کار می‌روند. داده‌های تعبیه شده و سایر داده‌های پردازش شده نیز در MinIO ذخیره می‌شوند تا در مراحل بعدی مورد استفاده قرار گیرند. برای ذخیره و بازیابی داده‌ها، از دو ابزار اصلی PostgreSQL و MinIO استفاده می‌شود. در کلاس پیش‌پردازش، فایل‌های تعبیه‌گر GloVe از MinIO دانلود و بردارهای تعبیه برای واژگان موجود در داده‌های



شکل ۵-۱۳: چرخه کاری آموزش مسئله تحلیل احساسات در بازار سهام

آموزش ساخته می‌شود. همچنین، داده‌های پردازش شده نیز در MinIO ذخیره می‌شوند. PostgreSQL نیز برای بازیابی مجموعه داده اولیه به کار می‌رود، به طوری که داده‌ها از پایگاهداده خوانده شده و برای پردازش و آماده‌سازی بیشتر به کلاس پیش‌پردازش منتقل می‌شوند.

در بخش آموزش از شبکه‌های حافظه کوتاه-مدت بلند^۹ به دلیل توانایی‌شان در درک و پردازش توالی‌های زمانی طولانی مدت، برای تحلیل احساسات متنی استفاده شده است. برخلاف مدل‌های سنتی که ممکن است نتوانند ارتباطات طولانی مدت بین کلمات را به خوبی درک کنند، LSTM با استفاده از سلول‌های حافظه و مکانیزم‌های دروازه‌ای خود، قادر است اطلاعات مهم را حفظ و اطلاعات غیرضروری را فراموش کند. برای استفاده از LSTM در تحلیل احساسات، از داده‌های پیش‌پردازش شده که در MinIO ذخیره‌سازی شده بود استفاده می‌شود. پس از آماده‌سازی داده‌ها، مدل LSTM با تعریف لایه‌های مختلف از جمله لایه‌های تعبیه‌گر، لایه‌های LSTM دوطرفه و لایه‌های چگال ساخته می‌شود. این لایه‌ها به مدل امکان می‌دهند تا وابستگی‌های معنایی و زمانی بین کلمات را بهتر درک کنند. علاوه بر این، به منظور جلوگیری از بیش‌پرازش از لایه‌های Dropout با نرخ 0.2 استفاده شده است.

در انتها پس از آموزش، مدل با استفاده از مجموعه دادگان ارزیابی، بررسی می‌شود. معیارهای ارزیابی مدل نیز دقت آن می‌باشد. در صورتی که بررسی‌های انجام شده از دقت مدل‌های گذشته بیشتر بود، مدل در MinIO ذخیره‌سازی می‌شود. به منظور اجرای چرخه یادگیری ماشین گفته شده این چرخه با استفاده از کتابخانه kfp روی بستر پلتفرم اعمال شده تا فرآیند پیش‌پردازش، آموزش و ارزیابی مدل را روی پلتفرم انجام دهد (شکل ۵-۱۳).

Long short-term memory (LSTM)^۹



شکل ۱۴-۵: خط لوله استقرار مسئله تحلیل احساسات در بازار سهام

خط لوله استقرار

همان‌طور که در شکل ۱۴-۵ مشاهده می‌کنید، اجرای خط لوله استنتاج مدل در پلتفرم با استفاده از Kserve شامل چند مرحله کلیدی است. ابتدا، یک فایل پیکربندی برای Kserve ایجاد می‌شود که جزئیات مدل، مسیر مدل ذخیره شده و هرگونه مراحل پیش‌پردازش یا پس‌پردازش لازم را مشخص می‌کند. سپس، یک جزء^{۱۰} Kserve با استفاده از کتابخانه kfp ایجاد می‌شود که از فایل پیکربندی YAML استفاده می‌کند. این جزء باید شامل منطق لازم برای استقرار مدل، مدیریت درخواست‌های ورودی و ارائه پیش‌بینی‌ها باشد. پس از ایجاد جزء، با استفاده از Kubeflow Pipeline روی بستر پلتفرم پیاده‌سازی می‌شود. این شامل تعریف وابستگی‌ها بین اجزاء است تا اطمینان حاصل شود که جزء استقرار مدل پس از اجزاء آموزش مدل و ارزیابی اجرا می‌شود. در نهایت، ارتباط با Endpoint API از طریق REST API انجام می‌شود. این فرآیند به تحلیلگران اجازه می‌دهد تا مدل‌های یادگیری ماشین را به طور مؤثر استقرار و مدیریت کنند و از ویژگی‌های پیشرفته مانند مقیاس‌پذیری خودکار، تعادل بار و پایش عملکرد بهره‌مند شوند.

نتایج

از آنجا که دقت و کارایی مدل هدف پیاده‌سازی مسئله نبوده و صرفاً هدف ما تست عملکردی پلتفرم می‌باشد، ملاک ارزیابی ما زمان اجرای خط‌لوله‌ها و چرخه یادگیری ماشین، مدت زمان پاسخ و میزان مصرف منابع می‌باشد. در جدول ۵-۵ زمان اجرای هر بخش را برای اجرای یک تغییر نشان داده شده است. کل مدت زمان آموزش تا استقرار مدل مجموعاً ۳۹۲ ثانیه می‌باشد. توجه داشته باشید که این زمان بدون لحاظ زمان کش می‌باشد. در صورتی که تغییر در یک جزء صورت می‌گیرد تنها آن جزء اجرا شده و بقیه اجزاء از حافظه نهان برای اجرا آن استفاده می‌کنند.

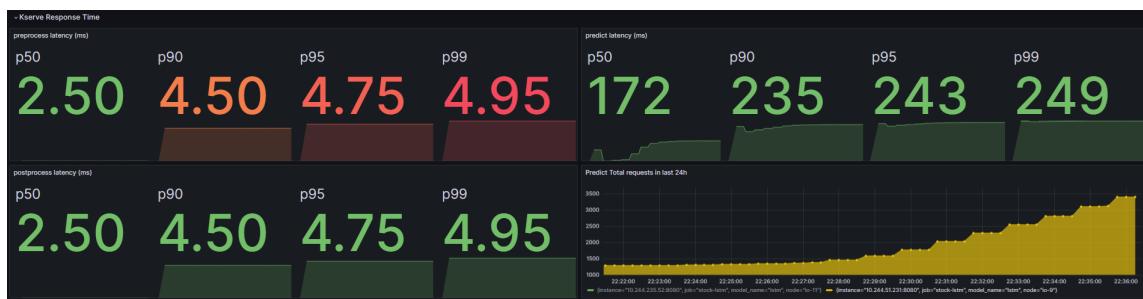
^{۱۰}Component

جدول ۵-۵: زمان اجرا در مسئله تحلیل احساسات در بازار سهام

خطوله	زمان (ثانیه)
آموزش CI/CD	70
چرخه کاری آموزش	187
استقرار CI/CD	37
چرخه کاری استقرار	98

جدول ۵-۶: تست استنتاج مدل مسئله تحلیل احساسات در بازار سهام

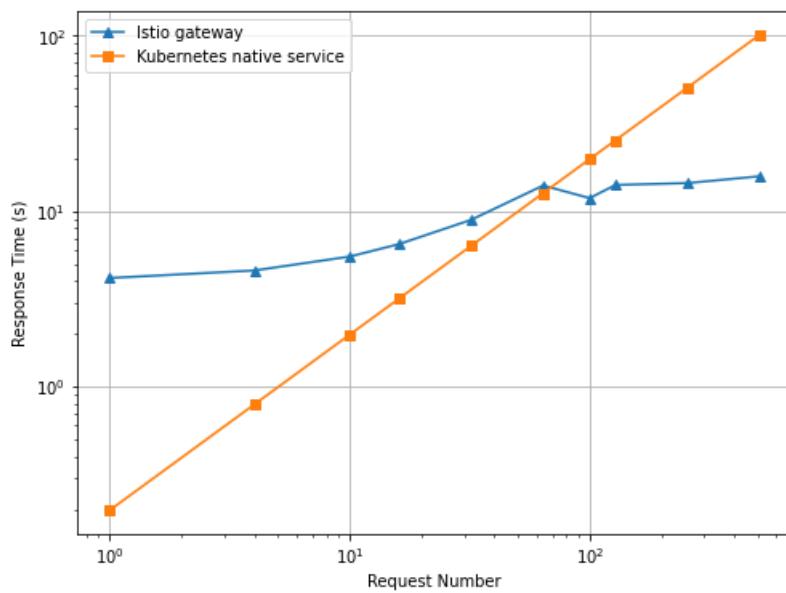
تعداد	زمان پاسخ (ثانیه)	CPU (mili cores)
1	4.16	24
4	4.59	110
10	5.51	290
16	6.48	387
32	8.91	566
64	13.99	845
100	11.86	1142
128	14.09	1338
256	14.45	1689
512	15.8	2090



شکل ۱۵-۵: گراف استنتاج مدل در مسئله تحلیل احساسات در بازار سهام (داشبورد Grafana)

به منظور تست مدل استقرار یافته، به صورت همزمان به آن درخواست داده و زمان پاسخ و میزان مصرف CPU را گزارش کردہ‌ایم. نتایج این تست در جدول ۶-۵ گزارش شده‌است. برای یک درخواست ارسال شده به مدل، مدت زمان استنتاج مدل تقریباً بین ۱۵۰ تا ۱۷۰ میلی‌ثانیه و بقیه زمان ارسال و دریافت درخواست از طریق Istio gateway می‌باشد (شکل ۱۵-۵).

نتایج در تعداد درخواست‌ها بالا به دلیل استفاده از تکنیک مقیاس‌پذیری خودکار در پلتفرم با استفاده از ابزارهای Istio و Knative مناسب است ولی در تعداد درخواست‌های پایین مناسب نمی‌باشد. به منظور رفع این مشکل از سرویس LoadBalancer در خوش کوبرنیتیز استفاده شده‌است که نتایج آن را در شکل ۱۶-۵ مشاهده می‌کنید. در این روش به دلیل



شکل ۵-۱۶: زمان استنتاچ مدل در مسئله تحلیل احساسات در بازار سهام

استفاده نکردن از تکنیک مقیاس‌پذیری خودکار، زمان درخواست‌ها در سناریوهایی که تعداد درخواست‌های همزمان زیاد می‌باشد، مناسب نمی‌باشد.

فصل ۶

نتیجه‌گیری و پیشنهادات

ما یک پلتفرم MLOps متن باز و مبتنی بر ابر طراحی کرده‌ایم که خودکارسازی و بازتولید را در اکثر مراحل فرآیند یادگیری ماشین فراهم می‌کند. این پلتفرم در محیط‌های کوبرنتیز ارائه‌دهنده مختلف ابری، کوبرنتیز داخلی و کوبرنتیز شبیه‌سازی شده روی ماشین‌های محلی اجرا می‌شود. هدف این طراحی، ایجاد زیرساختی است که مناسب اکثر پروژه‌ها و تیم‌های یادگیری ماشین باشد و فرآیندهای آن‌ها را به صورت خودکار و مقیاس‌پذیر نماید.

محدودیت‌های فعلی و کارهای آینده شامل چندین بهبود مهم است که باید در پلتفرم MLOps ما اعمال شود. پلتفرم برای اجرا به زیرساختی با منابع سخت‌افزاری زیاد نیاز دارد که توسعه محلی را دشوار و هزینه‌های سخت‌افزاری را افزایش می‌دهد؛ بنابراین، بهینه‌سازی‌های بیشتری برای سبک‌تر کردن پلتفرم ضروری است. بهینه‌سازی سرویس مش با جایگزینی Istio با یک سرویس مش سبک‌تر، ادغام استفاده از سرویس مش با سرویس‌های کوبرنتیز به منظور افزایش سرعت پاسخ دهی به درخواست کاربر و کاهش اجزای غیرضروری Knative-Serving نیز مورد نیاز است. امنیت سایبری پلتفرم باید تقویت شود و یک مطالعه موردنی برای ارزیابی عملکرد و قابلیت اجرایی پلتفرم با پیاده‌سازی سیستم‌های یادگیری ماشین پیچیده و پرمنع مانند آموزش مدل‌های زبانی بزرگ انجام شود. بهبودهای آینده شامل توسعه مرحله مدیریت شده برای برچسب‌گذاری داده‌ها، بهبودهای رابط کاربری برای ساده‌سازی استفاده از پلتفرم و ایجاد فرم‌های رابط کاربری برای افزودن استقرارها و جریان‌های کاری است.

مراجع

- [1] L Navarro E Hernanchez-sanchez F Rodriguez-Haro, F Freitag, “A summary of virtualization techniques,” .
- [2] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl, “Machine learning operations (mlops): Overview, definition, and architecture,” *IEEE Access*, vol. 11, pp. 31866–31879, 2023.
- [3] Algorithmia, “2020 state of enterprise machine learning,” 2020.
- [4] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer, “What is devops? a systematic mapping study on definitions and practices,” in *Proceedings of the Scientific Workshop Proceedings of XP2016*. 2016, Association for Computing Machinery.
- [5] Inc. Amazon Web Services, “What is devops?,” URL: <https://aws.amazon.com/devops/what-is-devops/> [Accessed: 2024-05-07].
- [6] Manish Virmani, “Understanding devops and bridging the gap from continuous integration to continuous delivery,” in *Fifth International Conference on the Innovative Computing Technology (INTECH 2015)*, 2015, pp. 78–82, doi: 10.1109/INTECH.2015.7173368.
- [7] A. Van Bennekum A. Cockburn-W. Cunningham M. Fowler J. Grenning J. Highsmith A. Hunt R. Jeffries K. Beck, M. Beedle, “Manifesto for agile software development,” 2001, URL: <https://agilemanifesto.org/> [Accessed: 2024-02-17].
- [8] Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley Professional, 2010.
- [9] paul hammant, “Trunk based development,” URL: <https://trunkbaseddevelopment.com/> [Accessed: 2023-11-01].
- [10] M. Huttermann, *DevOps for Developers*, chapter Infrastructure as Code, 2012.
- [11] Matej Artac, Tadej Borovssak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri, “Devops: introducing infrastructure-as-code,” in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, May 2017, pp. 497–498, doi: 10.1109/ICSE-C.2017.162.
- [12] Nelly Delgado, Ann Q Gates, and Steve Roach, “A taxonomy and catalog of runtime software-fault monitoring tools,” *IEEE Transactions on software Engineering*, vol. 30, no. 12, pp. 859–872, 2004.
- [13] SAIBS Arachchi and Indika Perera, “Continuous integration and continuous delivery pipeline automation for agile software project management,” in *Moratuwa Engineering Research Conference (MERCon)*, May 2018, pp. 156–161, doi: 10.1109/MERCon.2018.8421965.
- [14] Anja Kammer Florian Beetz and Dr. Simon Harrer, *GitOps Cloud-native Continuous Deployment*, 2021.
- [15] N. Forsgren and J. Humble, “The role of continuous delivery in it and organizational performance,” March 2016, doi: 10.2139/ssrn.2681909.
- [16] Inc. Amazon Web Services, “What is virtualization?,” URL: <https://aws.amazon.com/what-is/virtualization/> [Accessed: 2024-05-08].

- [17] Amit M Potdar, DG Narayan, Shivaraj Kengond, and Mohammed Moin Mulla, “Performance evaluation of docker container and virtual machine,” *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020.
- [18] International Business Machines (IBM) Inc., “Containerization,” URL: <https://www.ibm.com/topics/containerization> [Accessed: 2023-05-21].
- [19] Docker Inc., “Docker,” URL: <https://docs.docker.com/> [Accessed: 2023-05-18].
- [20] Y. Yu Y. Zhou and B. Ding, “Towards mlops: A case study of ml pipeline platform,” October 2020, doi: 10.1109/ICAICE51518.2020.00102.
- [21] Damian A. Tamburri, “Sustainable mlops: Trends and challenges,” in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 2020, pp. 17–23, doi: 10.1109/SYNASC51798.2020.00015.
- [22] Lucy Ellen Lwakatare, Ivica Crnkovic, Ellinor Rånge, and Jan Bosch, “From a data science driven process to a continuous delivery process for machine learning systems,” in *Product-Focused Software Process Improvement*, Maurizio Morisio, Marco Torchiano, and Andreas Jedlitschka, Eds., Cham, 2020, pp. 185–201, Springer International Publishing.
- [23] Ioannis Karamitosos, Saeed Albarhami, and Charalampos Apostolopoulos, “Applying devops practices of continuous automation for machine learning,” *Information*, vol. 11, no. 7, 2020, doi: 10.3390/info11070363.
- [24] Lucas Cardoso Silva, Fernando Rezende Zagatti, Bruno Silva Sette, Lucas Nildaimon dos Santos Silva, Daniel Lucrédio, Diego Furtado Silva, and Helena de Medeiros Caseli, “Benchmarking machine learning solutions in production,” in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020, pp. 626–633, doi: 10.1109/ICMLA51294.2020.00104.
- [25] Behrouz Derakhshan, Alireza Rezaei Mahdiraji, Tilmann Rabl, and Volker Markl, “Continuous deployment of machine learning pipelines,” in *EDBT*, March 2019, pp. 397–408, doi: 10.5441/002/edbt.2019.35.
- [26] Alexandra Posoldova, “Machine learning pipelines: From research to production,” *IEEE Potentials*, vol. 39, no. 6, pp. 38–42, 2020, doi: 10.1109/MPOT.2020.3016280.
- [27] Airflow, “Apache airflow,” URL: <https://airflow.apache.org/> [Accessed: 2023-11-02].
- [28] Kubeflow, “Kubeflow,” URL: <https://www.kubeflow.org/> [Accessed: 2023-11-02].
- [29] M. Schmitt, “Airflow vs. luigi vs. argo vs. mlflow vs. kubeflow,” URL: <https://www.datarevenue.com/en-blog/airflow-vs-luigi-vs-argo-vs-mlflow-vs-kubeflow> [Accessed: 2023-11-02].
- [30] Feast, “Feast,” URL: <https://feast.dev/> [Accessed: 2023-11-05].
- [31] Álvaro López García, Jesús Marco De Lucas, Marica Antonacci, Wolfgang Zu Castell, Mario David, Marcus Hardt, Lara Lloret Iglesias, Germán Molto, Marcin Plociennik, Viet Tran, Andy S. Alic, Miguel Caballer, Isabel Campos Plasencia, Alessandro Costantini, Stefan Dlugolinsky, Doina Cristina Duma, Giacinto Donvito, Jorge Gomes, Ignacio Heredia Cacha, Keiichi Ito, Valentin Y. Kozlov, Giang Nguyen, Pablo Orviz Fernández, Zdenek Šustr, and Pawel Wolniewicz, “A cloud-based framework for machine learning workloads and applications,” *IEEE Access*, vol. 8, pp. 18681–18692, 2020, doi: 10.1109/ACCESS.2020.2964386.
- [32] MLflow Pro, “MLflow,” URL: <https://mlflow.org/> [Accessed: 2023-11-21].
- [33] Apache Spark, “Apache spark,” URL: <https://spark.apache.org/> [Accessed: 2023-11-29].
- [34] Apache Hadoop, “Apache hadoop,” URL: <https://hadoop.apache.org/> [Accessed: 2023-11-29].
- [35] Knative, “Knative,” URL: <https://knative.dev/docs/> [Accessed: 2023-11-29].
- [36] Cédric Renggli, Luka Rimanic, Nezihe Merve Gürel, Bojan Karlas, Wentao Wu, and Ce Zhang, “A data quality-driven view of mlops,” *CoRR*, vol. abs/2102.07750, 2021, URL: <https://arxiv.org/abs/2102.07750> [Accessed: 2023-11-30].
- [37] GitLab Inc, “Gitlab,” URL: <https://about.gitlab.com/> [Accessed: 2024-05-16].

- [38] Gerrit, “Gerritcodereview,” URL: <https://www.gerritcodereview.com/> [Accessed: 2024-05-16].
- [39] Jenkins, “Jenkins,” URL: <https://www.jenkins.io/> [Accessed: 2023-11-01].
- [40] Jupyter, “Jupyter,” URL: <https://jupyter.org/> [Accessed: 2023-11-31].
- [41] Tao Cui, Ye Wang, and Bassel Namih, “Build an intelligent online marketing system: An overview,” *IEEE Internet Computing*, vol. 23, no. 4, pp. 53–60, 2019.
- [42] Red Hat Inc., “Ansible,” URL: <https://www.ansible.com/> [Accessed: 2023-12-01].
- [43] HashiCorp Inc., “Terraform,” URL: <https://www.terraform.io/> [Accessed: 2023-12-01].
- [44] M. Luksa, *Kubernetes in Action*, Manning, 2018.
- [45] K. Hightower, B. Burns, and J. Beda, *Kubernetes: Up and Running: Dive into the Future of Infrastructure*, O'Reilly Media, 2017.
- [46] MinIO Inc., “Minio,” URL: <https://min.io/> [Accessed: 2023-12-05].
- [47] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han, “Service mesh: Challenges, state of the art, and future research opportunities,” in *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, 2019, pp. 122–1225.
- [48] L. Calcote and Z. Butcher, *Istio: Up and Running: Using a Service Mesh to Connect, Secure, Control, and Observe*, O'Reilly Media, 2019.
- [49] Daniel Fett, Ralf Küsters, and Guido Schmitz, “The web sso standard openid connect: In-depth formal security analysis and security guidelines,” in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 189–202.
- [50] J. Pivotto and B. Brazil, *Prometheus: Up & Running*, O'Reilly Media, 2023.
- [51] Raintank Inc., “Grafana,” URL: <https://grafana.com/> [Accessed: 2023-12-29].
- [52] L.T. Grant, *Kubeflow for Machine Learning: From Lab to Production*, O'Reilly Media, Incorporated, 2021.
- [53] The KServe, “Kserve,” URL: <https://kserve.github.io/> [Accessed: 2024-1-21].

ABSTRACT

In the current era, artificial intelligence and machine learning have become vital and widely used technologies across various industries. These technologies enable companies and organizations to optimize processes, predict trends, and uncover hidden patterns in data with high accuracy and speed. However, fully leveraging the capabilities of AI and ML requires the effective and efficient deployment of ML models in production environments. MLOps, a combination of DevOps and ML concepts, aids in managing the lifecycle of ML models from development to deployment and maintenance. In this research, due to international sanctions and limited access to external services such as Google Vertex AI, Amazon SageMaker, and Microsoft Azure ML, a native MLOps platform has been designed and implemented. This platform is developed using open-source tools and provides various features including lifecycle management, performance monitoring, automatic updates, and integration with existing tools. The research aims to address questions related to the needs of a modern MLOps platform and the feasibility of its implementation using open-source tools. The results from the implementation and tests indicate that the proposed platform not only meets modern requirements but also ensures independence from cloud providers. Finally, the findings and recommendations for future research are presented.

KEYWORDS

1. Machine Learning Operation (MLOps).
2. Development and Operation (DevOps).
3. Continuous Integration (CI).
4. Continuous Development (CD).
5. Machine Learning Lifecycle Management.
6. Open-Source MLOps Platform.



SHARIF UNIVERSITY OF TECHNOLOGY
ELECTRICAL ENGINEERING DEPARTMENT

M.Sc. THESIS

Title:

Design and Implementation of GPU-based MLOps Cloud Platform

by:

Abolfazl Yarian

Supervisor:

Dr. Matin Hashemi

July 2024