



دانشگاه صنعتی شریف

دانشکده مهندسی برق

پایان نامه کارشناسی ارشد  
گرایش سیستم های الکترونیک دیجیتال

# پیاده سازی یک پلتفرم MLOps به صورت ابری روی GPU

نگارنده

ابوالفضل یاریان

استاد راهنما

دکتر متین هاشمی

خردادماه ۱۴۰۳



### توجه

این پروژه بر اساس قرارداد شماره (.....) از حمایت مالی  
مرکز تحقیقات مخابرات ایران برخوردار شده است.

بسمه تعالی

دانشگاه صنعتی شریف  
دانشکده مهندسی برق

پایان نامه کارشناسی ارشد

عنوان: مدلسازی نهان نگاری تصویر بر اساس تئوری اطلاعات

نگارش: «نام و نام خانوادگی دانشجو»

اعضا هیات داوران:

.....امضاء:	دکتر ...
.....امضاء:	دکتر ...
.....امضاء:	دکتر ...
.....امضاء:	دکتر ...
.....امضاء:	دکتر ...

تاریخ: ۶ شهریور ۱۳۸۴.

## تقدیم و قدردانی

در این صفحه از کسانی که مایلید تشکر می‌کنید.

## چکیده:

در دنیای دیجیتال امروزه، نهان نگاری مقاوم تصویر که در آن یک سیگنال حامل داده به صورت نامرئی و مقاوم در برابر حملات در تصویر تعبیه می‌شود، به عنوان یک راهکار برای حل مساله حفاظت از حق تالیف محصولات تصویری معرفی شده است. برای این منظور تاکنون جهت نهان نگاری روشهای متعددی به کار گرفته شده است که از آن جمله می‌توان به استفاده از مدل‌های بینایی جهت یافتن میزان بیشینه انرژی نهان نگاره برای تعبیه در تصویر و استفاده از حوزه های مقاوم در برابر حملات، اشاره نمود. در همین راستا در این پایان نامه به استفاده از مفاهیم حوزه تئوری اطلاعات به عنوان یک راهنما در توسعه الگوریتم‌های موجود، جهت قرار دادن بهینه نهان نگاره پرداخته شده است. همچنین در ساختار پیشنهادی که برای افزایش مقاومت در حوزه تبدیل تصویر پیاده می‌شود، از تبدیلات چنددقتی مانند تبدیل موجک گسسته و تبدیل MR-SVD که به سیستم بینایی انسان نزدیک‌ترند، استفاده می‌شود. به طوریکه در حوزه تبدیل موجک، با استفاده از آنتروپی و تاثیر پدیده پوشش آنتروپی به اصلاح مدل‌های بینایی مرتبط با این حوزه پرداخته و بدین ترتیب نهان نگاره با قدرت و مقاومت بالاتر در تصویر تعبیه نموده و همچنین کیفیت بهتر برای تصویر نهان نگاری شده بدست آمد. همچنین در حوزه تبدیل MR-SVD ابتدا این تبدیل که تاکنون برای نهان نگاری استفاده نشده بود، جهت نهان نگاری بکار گرفته شد و سپس مشابه ساختار پیشنهادی مبتنی بر آنتروپی در حوزه تبدیل موجک، در حوزه این تبدیل نیز بکار رفت و نتایج شبیه‌سازیها مقاوم‌تر بودن ساختار پیشنهادی و کیفیت بالاتر تصویر نهان نگاری شده در این حوزه را نتیجه داد.

## کلمات کلیدی:

- |                       |                                    |
|-----------------------|------------------------------------|
| ۱- نهان نگاری تصویر   | Image Watermarking                 |
| ۲- تبدیل چنددقتی      | Multi-Resolution Transform         |
| ۳- سیستم بینایی انسان | Human Visual System (HVS)          |
| ۴- تبدیل موجک         | Wavelet Transform                  |
| ۵- تجزیه مقادیر تکین  | Singular Value Decomposition (SVD) |
| ۶- آنتروپی            | Entropy                            |
| ۷- پوشش آنتروپی       | Entropy Masking                    |

# فهرست مطالب

۱	مقدمه	۱
۲	مرور مفاهیم پایه	۲
۲	۱-۲ DevOps	۲
۲	۱-۱-۲ تعریف	۲
۳	۲-۱-۲ چرخه کاری DevOps	۳
۴	۳-۱-۲ خط لوله CI/CD	۴
۷	۴-۱-۲ مزایای متدولوژی DevOps	۷
۹	۲-۲ مجازی سازی و کانتینرها	۹
۹	۱-۲-۲ مجازی سازی	۹
۱۱	۲-۲-۲ کانتینرها	۱۱
۱۳	۳-۲-۲ هماهنگ سازی کانتینرها (کوپرنیتیز)	۱۳
۱۶	MLOps	۳
۱۶	۱-۳ مقدمه	۱۶
۱۶	۲-۳ تعریف مفاهیم اولیه	۱۶
۱۷	۱-۲-۳ اصول	۱۷
۱۹	۲-۲-۳ اجزاء	۱۹
۲۵	۳-۲-۳ نقش ها	۲۵
۲۶	۳-۳ معماری جامع	۲۶
۳۴	طراحی یک پلتفرم MLOps	۴

۳۴	.....	مقدمه	۱-۴
۳۴	.....	سیستم مدیریت پلتفرم	۲-۴
۳۴	.....	مدیریت پیکربندی و فراهم سازی زیرساخت	۱-۲-۴
۳۶	.....	خط لوله CI/CD	۲-۲-۴
۳۸	.....	مخزن کد منبع	۳-۲-۴
۳۹	.....	مخزن مؤلفه ها	۴-۲-۴
۴۰	.....	معماری خوشه کوبرنتیز	۳-۴
۴۱	.....	مدیریت داده	۱-۳-۴
۴۴	.....	شبکه	۲-۳-۴
۴۷	.....	مدیریت کاربران و چندمستاجری	۳-۳-۴
۴۹	.....	نظارت	۴-۳-۴
۵۰	.....	استقرار مدل	۵-۳-۴

## ۵ پیاده سازی و نتایج

۵۱	.....	پیاده سازی پلتفرم	۱-۵
۵۱	.....	سیستم مدیریت	۱-۱-۵
۵۴	.....	خوشه کوبرنتیز	۲-۱-۵
۵۶	.....	حل مسئله و نتایج	۲-۵

## ۶ نتیجه گیری و پیشنهادات



# فهرست جداول

۱-۲	نمونه هایی از ابزار برای مراحل خاص اتوماسیون خط لوله CI/CD	۵
۱-۵	مشخصات سخت افزاری ماشین های مدیریت	۵۲
۲-۵	مشخصات سخت افزاری ماشین های خوشه کوبرنتیز	۵۴

# فهرست تصاویر

۴	..... مراحل DevOps	۱-۲
۱۰	..... انواع هایپروایزر [۱]	۲-۲
۱۱	..... تفاوت ماشین مجازی و کانتینر	۳-۲
۱۳	..... معماری لایه ای تصویر داکر	۴-۲
۱۴	..... مولفه های یک خوشه کوبرنتیز	۵-۲
۲۰	..... خط لوله در Apache Airflow	۱-۳
۲۱	..... انباره ویژگی	۲-۳
۲۵	..... نقش ها و اشتراکات آنها در پارادایم MLOps	۳-۳
۲۶	..... معماری جامع MLOps	۴-۳
۳۵	..... معماری Ansible	۱-۴
۳۷	..... استقرار مبتنی بر Pull	۲-۴
۳۷	..... استقرار مبتنی بر Push	۳-۴
۴۲	..... نحوه کار PV و PVC در خوشه کوبرنتیز	۴-۴
۴۳	..... معماری MinIO	۵-۴
۴۶	..... معماری Istio	۶-۴
۴۸	..... روند احراز هویت	۷-۴
۵۳	..... مخازن مولفه در Nexus	۱-۵
۵۳	..... خط لوله ها CI/CD در Jenkins	۲-۵
۵۳	..... مخازن کد در Gitlab	۳-۵
۵۵	..... میزان مصرف منابع سخت افزاری پلتفرم بدون بار	۴-۵

# فهرست کلمات اختصاری

2D-DWT	2-Dimensional Discrete Wavelet Transform
CPD	Cycle Per Degree
CSF	Contrast Sensitivity Function
⋮	⋮

## فصل ۱

### مقدمه

گسترش روز افزون شبکه جهانی اینترنت و توسعه فناوری اطلاعات، نیاز فزاینده‌ای را به استفاده از سرویسهای چندرسانه‌ای دیجیتال، در پی داشته به طوریکه کاربردهای دیجیتال شاهد رشد شگرفی در طول دهه گذشته بوده است که نتیجه آن ایجاد سیستمهای کارآمد در ذخیره، انتقال و بازیابی اطلاعات است. مزایای فراوان فناوری دیجیتال، باعث محبوبیت و کاربرد هر چه بیشتر آن توسط اشخاص شده تا جاییکه حتی وسایل ضبط و پخش صدا و تصویر آنالوگ خانگی هم به سرعت با نمونه‌های دیجیتال جایگزین شده‌اند. اما این موضوع مسائل حاشیه‌ای دیگری برای بشر ایجاد نموده است. به طوریکه امکان تهیه کپی‌های متعدد از روی نسخه اصلی بدون کاهش کیفیت آن و یا سادگی جعل و تغییر محتوای اطلاعاتی نسخه اصلی، باعث شده که مالکیت معنوی<sup>۱</sup> صاحبان اثر به خطر افتاده و در نتیجه بسیاری از ارائه دهندگان سرویسهای چندرسانه‌ای (از جمله شرکتهای فیلم‌سازی) از ارائه نمونه دیجیتال محصولاتشان خودداری نمایند. لذا برطرف نمودن این مشکلات، یکی از زمینه‌های پژوهشی مهم در عرصه مخابرات و بخصوص پردازش سیگنال است.

---

<sup>۱</sup> Intellectual Property

## فصل ۲

# مرور مفاهیم پایه

## ۱-۲ DevOps

### ۱-۱-۲ تعریف

دِوایس که از اتحاد واژگان Development و Operation به وجود آمده است؛ ترکیبی از ابزارها، کنش‌ها و فرهنگ کاری است که تیم‌های توسعه<sup>۱</sup> و عملیات<sup>۲</sup> را به همکاری موثرتر نزدیک می‌کند و کسب و کارها با استفاده از می‌توانند اپلیکیشن‌ها و سرویس‌هایشان را با سرعت بالاتری نسبت به روش‌های سنتی تحویل دهند. همین سریع‌تر شدن سرعت توسعه و انتشار نرم‌افزار، سازمان‌ها را قادر می‌سازد تا در مقایسه با کسب‌وکارهایی که هنوز از روش‌های سنتی توسعه نرم‌افزار استفاده می‌کنند خدمات بهتری به مشتریان ارائه دهند. در واقع دِوایس سعی دارد تا مشکل جدایی تیم‌های مختلف را رفع کرده و یک فرهنگ سازمانی یکپارچه را میان تیم‌های مختلفی که در حال توسعه یک نرم‌افزار هستند ایجاد کند. از این جهت بسیاری از کارها می‌تواند به صورت خودکار پیش رفته و در نهایت همه چیز با سرعت بیشتری صورت بگیرد [۲، ۳]. این خودکار سازی با استفاده از خط لوله CI/CD از منبع کد شروع می‌شود و تا مانیتورینگ محصول ادامه میابد [۴].

تا قبل از تشکیل دِوایس، تیم‌های توسعه نرم‌افزار یا تیم عملیاتی در محیط‌های جداگانه کار می‌کردند. هدف تیم توسعه تولید محصول جدید و با افزودن ویژگی‌های جدیدی روی محصولات قبلی بود. هدف تیم عملیاتی نیز ثابت نگه داشتن وضعیت موجود سرویس‌ها برای پایداری بیشتر بود. به مرور زمان در فرآیند توسعه نرم‌افزار، روش‌های چابک<sup>۳</sup> ایجاد شد تا با مشتری تعامل بهتری برقرار شود و نیازهایی که دارد به محصول اضافه شود [۵]. جدایی دو تیم توسعه و عملیات از هم باعث

---

Development<sup>۱</sup>  
Operation<sup>۲</sup>  
Agile<sup>۳</sup>

می‌شد که در فرآیند تولید محصول و استقرار<sup>۴</sup> آن، اتلاف وقت ایجاد شود و محصول دیرتر به دست مشتری برسد [۶].

## ۲-۱-۲ چرخه کاری DevOps

همانطور که در شکل ۲-۱ مشاهده می‌کنید، DevOps قصد دارد از ابزار و جریان‌های کاری<sup>۵</sup> برای خودکارسازی یک یا چند مورد از موارد زیر استفاده کند:

۱. کدنویسی: شامل توسعه، بازبینی کد و ابزارهای کنترل نسخه است. مثلاً، یک تیم تصمیم می‌گیرد از گیت<sup>۶</sup> به عنوان ابزار کنترل نسخه و از گیت هاب<sup>۷</sup> نیز به عنوان یک مخزن راه دور استفاده کند. این تیم مجموعه‌ای از دستورالعمل‌های سبک کدنویسی را با استفاده از ابزاری نظیر Linter به همراه حداقل درصد پوشش تست تعریف کرده و با تعیین استراتژی انشعاب مبتنی بر تنه<sup>۸</sup> تغییرات خود را به منظور بازبینی برای ادغام با انشعاب اصلی<sup>۹</sup> برای توسعه دهنده ارشد ارسال می‌کند [۷].

۲. ساخت: شامل ایجاد و ذخیره خودکار مولفه<sup>۱۰</sup> ها می‌باشد. به طور مثال یک تیم تصمیم می‌گیرد یک Container image قابل اجرا از محصول خود ایجاد کند.

۳. تست: شامل ابزارهایی برای تست محصول می‌باشد. تیم محیطی را به منظور تست هر تغییر جدید راه اندازی می‌کند که در آن مجموعه‌ای از آزمایش‌ها مانند آزمون واحد<sup>۱۱</sup>، آزمون یکپارچگی<sup>۱۲</sup> و ... به طور خودکار در برابر 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 بازتولید محیط در حال اجرا و زیرساخت نرم افزاری شامل سیستم عامل تا پایگاه داده و سرویس‌های خاص و پیکربندی شبکه آنها می‌باشد [۸، ۹].

۵. استقرار: این مرحله شامل استراتژی استقرار است. به طور مثال تیم می‌تواند تصمیم بگیرد که یک محصول به طور

---

<sup>۴</sup> Deploy

<sup>۵</sup> Workflow

<sup>۶</sup> Git

<sup>۷</sup> Github

<sup>۸</sup> Trunk-Based

<sup>۹</sup> Merge request

<sup>۱۰</sup> Artifact

<sup>۱۱</sup> Unit test

<sup>۱۲</sup> Integration test



شکل ۲-۱: مراحل DevOps

مستقیم منتشر شود یا ابتدا در یک محیط آزمایشی مورد ارزیابی قرار گیرد. هم چنین در مواقعی که مشکلی در استقرار وجود دارد چه کاری انجام دهند و استراتژی بازگشت<sup>۱۳</sup> خود را پیاده سازی کنند.

۶. نظارت: از عملکرد محصول تا نظارت بر تجربه کاربر نهایی را شامل می شود. به عنوان مثال، می تواند مدت زمان درخواست های پایگاه داده یا بارگذاری وبسایت یا تعداد کاربرانی که از ویژگی های خاص محصول استفاده می کنند یا تعداد بازدیدکنندگان از یک وبسایت که به ثبت نام ختم می شود یا تعداد کاربران جدید در یک مجموعه زمانی خاص را پوشش دهد. مرحله نظارت هم چنین شامل هشدار خودکار خرابی ها نیز می باشد (به عنوان مثال، آستانه استفاده از CPU [۱۰]). در نهایت نظارت بر محیط تولید به منظور اطمینان از صحت کارکرد صحیح محصول ضروری است.

## ۳-۱-۲ خط لوله CI/CD

در دنیای توسعه نرم افزار دستیابی به بهره وری بالا، کیفیت مطلوب محصول و رضایت مشتری از اهداف اصلی هر سازمانی است. در متدولوژی DevOps و رویکردهای مرتبط با آن مانند ادغام مداوم<sup>۱۴</sup>، تحویل مداوم<sup>۱۵</sup> و استقرار مداوم<sup>۱۶</sup> به طور فزاینده ای محبوب شده اند زیرا به سازمان ها کمک می کنند تا با سرعت و کارآمدی بیشتری به این اهداف دست یابند [۶].

ادغام مداوم به فرآیندی اطلاق می شود که در آن توسعه دهندگان برنامه های خود را به طور مداوم (معمولاً چندین بار در روز) در یک مخزن مشترک ادغام می کنند. به محض ادغام کد، یک سری از تست های خودکار اجرا می شود تا اطمینان حاصل شود که این تغییرات جدید باعث بروز مشکل در نرم افزار نشده اند [۱۱]. این تست ها شامل تست های واحد، تست های یکپارچگی و تست های کارکردی می باشند. از آنجایی که برنامه های کاربردی پیشرفته کنونی در چندین پلتفرم و ابزار های مختلف اقدام به توسعه می کنند، لذا نیاز به مکانیزمی برای ادغام و تایید تغییرات مختلف، اهمیت بالاتری پیدا می کند.

<sup>۱۳</sup> Rollback<sup>۱۴</sup> Continuous Integration (CI)<sup>۱۵</sup> Continuous Delivery (CD)<sup>۱۶</sup> Continuous Deployment (CD)

جدول ۱-۲: نمونه هایی 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

**تحويل مداوم** ادامه ای بر ادغام مداوم است و به تیم‌ها این امکان را می‌دهد تا نرم‌افزار را پس 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 [۱۱]. به عبارتی دیگر، هدف از تحويل 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 تحويل 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 [۱۱]. این به معنای آن است که نسخه‌های جدید نرم‌افزار می‌توانند به طور مداوم و بدون دخالت دستی به کاربران نهایی تحويل 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 [۶]، با یک فرآیند ساخت شروع می‌شود. در این مرحله کد توسط ابزارهای مرتبط که در جدول ۱-۲ ذکر شده است، تبدیل به نرم‌افزار قابل اجرا می‌شوند. 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 نرم‌افزار نمی‌شوند. در صورت موفقیت‌آمیز بودن تست‌ها، یک نسخه قابل اجرا از کد نسخه گذاری شده و در مخازنی همانند Nexus نگه داری می‌شوند. در مرحله پیکربندی تنظیم محیط لازم برای نصب و استفاده از نرم‌افزار انجام می‌شود. دو رویکرد اصلی برای این کار وجود دارد: مرحله به مرحله<sup>۱۷</sup> و اعلامی<sup>۱۸</sup>. 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

Procedural<sup>۱۷</sup>Declarative<sup>۱۸</sup>



تعریف شده 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 [۱۲]. پس از این، مرحله‌ی استقرار آغاز می‌شود که در آن نرم‌افزار به محیط‌های تست، توسعه یا تولید منتقل می‌شود. این فرایند 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 Gitlab CI، CircleCI و Jenkins می‌تواند انجام گردد. در آخر نیز مرحله نظارت انجام می‌شود. ابزارهای نظارتی نظیر Prometheus و Grafana 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. هم چنین اطلاعاتی مانند زمان طول کشیده برای هر مرحله، تعداد خطاها و متوسط زمان بین خرابی‌ها<sup>۱۹</sup> 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 استاندارد های تیم توسعه در ابتدا خط لوله بخشی را قرار داد تا از استاندارد بودن کد اطمینان یابد. در این بخش می‌توان از Linter ها یا pre-commit hooks استفاده کرد. معروف ترین ابزار برای هر مرحله را در جدول ۲-۱ نیز مشاهده می‌کنید.

به هنگام طراحی و پیاده سازی یک خط لوله CI/CD باید به نکات زیر توجه کرد [۶، ۱۲، ۱۱]:

- قابلیت بازگشت به حالت قبل<sup>۲۰</sup>

- قابلیت مشاهده<sup>۲۱</sup> و هشدار دادن<sup>۲۲</sup>

- 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 قدیمی تر 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. هم چنین 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

<sup>۱۹</sup> Mean time between failures (MTBF)

<sup>۲۰</sup> Rollback

<sup>۲۱</sup> Observability

<sup>۲۲</sup> Alerting

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 کمی 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<sup>۲۳</sup> 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 متمرکز مانند مخازن کد منبع است. هم چنین مدیریت اسرار<sup>۲۴</sup> 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 باشد. بهبود و بهینه‌سازی این فرآیند از طریق اجرای موازی 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.

## ۴-۱-۲ مزایای متدولوژی DevOps

این متدولوژی یک رویکرد نوآورانه در توسعه نرم‌افزار و عملیات است که مزایای بسیاری برای بهبود عملکرد سازمانی<sup>۲۵</sup> ارائه می‌دهد [۱۳]. 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 [۳].

۱. افزایش سرعت و کارایی: با خودکارسازی فرایند انتشار نرم‌افزار از طریق CI/CD، تیم‌ها می‌توانند فرکانس و سرعت

<sup>۲۳</sup>Role-Based Access Control (RBAC)

<sup>۲۴</sup>Secrets

<sup>۲۵</sup>Organization performance

انتشارها را افزایش داده که منجر افزایش سرعت پاسخ دهی به مشتری شده و مزیت رقابتی ایجاد می کند [۴].

۲. ایجاد محیط‌های عملیاتی پایدارتر: تضمین قابلیت اطمینان به‌روزرسانی‌های برنامه و تغییرات زیرساخت یکی از مزایای مهم این متدلوژی می باشد. از طریق خط لوله CI/CD، هر تغییری برای اطمینان از کارایی و ایمنی ادغام با محیط تولید آزمایش می‌شود تا از انتشار نسخه‌های معیوب جلوگیری کند. یکی از شاخص‌های اصلی پایداری، انتشارهای متناوب و مکرر است. با استفاده از این متدلوژی توسعه‌دهندگان می‌توانند خطاها را سریع‌تر شناسایی و رفع کنند. این موضوع باعث کاهش شاخص  $MTTR^{26}$  می‌شود. این شاخص مدت زمان برگشت به وضعیت پایدار بعد از وقوع خطا یا اشکال را نشان می‌دهد و هرچه مقدار آن کمتر باشد، پایداری سیستم بیشتر است [۶]. 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 خطاها و در نتیجه حفظ پایداری دارند.

۳. مقیاس پذیری: تسهیل‌کننده مدیریت مقیاس‌پذیر زیرساخت‌ها و فرآیندهای توسعه است. تکنیک‌هایی مانند زیرساخت به عنوان کد<sup>۲۷</sup> مدیریت محیط‌های توسعه، آزمایش و تولید را به شکلی تکرارپذیر و کارآمد ساده‌سازی می‌کنند [۳].

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 متدلوژی و خط لوله CI/CD طول چرخه‌ها کوتاه‌تر و نتایج کمی و کیفی بهتر 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. چابکی در این متدلوژی اهمیت زیادی دارد لذا فناوری ابری نیز این چابکی را به تیم‌ها ارائه و سرعت و هماهنگی بین تیم‌ها را افزایش می‌دهد. با کمک این فناوری، حتی اگر در فرایند 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 پروژه، بهبود امنیت، افزایش مشارکت و بهبود دسترسی و کاربری داده‌ها اشاره کرد.

۵. تجزیه 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 متدلوژی و وجود همکاری فعال در تیم‌ها، ارتباطات بهبود می‌یابد. این امر باعث می‌شود که

<sup>26</sup>Mean Time To Recover  
<sup>27</sup>infrastructure as a code

اطلاعات به طور موثرتر جریان یابد، کارایی تیم‌ها افزایش یابد و در نتیجه، کارایی کلی سازمان بهبود پیدا کند.

## ۲-۲ مجازی سازی و کانتینرها

### ۱-۲-۲ مجازی سازی

تکنولوژی مجازی سازی<sup>۲۸</sup> به روشی اشاره دارد که در آن منابع سخت افزاری 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، نرم افزار و شبکه، که هر کدام کاربردهای خاص خود را دارند [۱، ۱۴].

مجازی سازی سرور یکی از تکنولوژی‌های کلیدی در مدیریت و بهره‌برداری از داده‌ها و منابع سخت افزاری در مراکز داده است. این فناوری امکان تقسیم یک سرور فیزیکی<sup>۲۹</sup> 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<sup>۳۰</sup>
- ماشین مجازی
- سیستم مدیریت مرکزی

هایپروایزر، که گاهی اوقات به عنوان مدیر ماشین مجازی<sup>۳۱</sup> شناخته می‌شود، نقش محوری در مجازی سازی سرور دارد. این نرم افزار بر روی سخت افزار سرور نصب می‌شود و وظیفه آن تقسیم منابع سرور فیزیکی، مانند CPU، حافظه، فضای دیسک و شبکه به چندین ماشین مجازی است. هایپروایزرها به دو دسته تقسیم می‌شوند.

هایپروایزر نوع<sup>۳۲</sup> مستقیماً بر روی سخت افزار نصب می‌شود و به طور مستقل از سیستم عامل فیزیکی عمل می‌کند. می‌توان از هایپروایزرهای نوع ۱ معروف به VMware ESXi، Microsoft Hyper-V و KVM اشاره کرد که برای بهینه سازی عملکرد و امنیت طراحی شده‌اند.

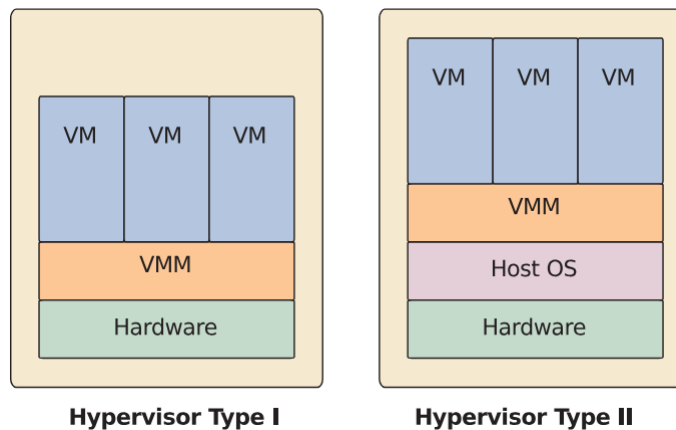
<sup>۲۸</sup>Virtualization

<sup>۲۹</sup>Bare-metal

<sup>۳۰</sup>Hypervisor

<sup>۳۱</sup>Virtual Machine Manager (VMM)

<sup>۳۲</sup>Bare-metal



شکل ۲-۲: انواع هایپروایزر [۱]

هایپروایزر نوع ۳۲ روی یک سیستم عامل میزبان نصب می‌شود و به عنوان یک برنامه درون سیستم عامل عمل می‌کند. از هایپروایزر نوع ۲ نیز می‌توان به VMware Workstation و Oracle VirtualBox اشاره کرد. این هایپروایزرها اغلب برای تست و توسعه مورد استفاده قرار می‌گیرند. در شکل ۲-۲ ساختار آن را مشاهده می‌کنید.

هایپروایزرها به لحاظ انعطاف پذیری 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<sup>۳۴</sup> و بازیابی فاجعه<sup>۳۵</sup> را ارائه دهند.

ماشین مجازی<sup>۳۶</sup> 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.

<sup>۳۲</sup>Hosted

<sup>۳۴</sup>Replication

<sup>۳۵</sup>Disaster Recovery

<sup>۳۶</sup>Virtual Machine (VM)



شکل ۲-۳: تفاوت ماشین مجازی و کانتینر

## ۲-۲-۲ 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 کنید، 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 نباشند.

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 سیستم‌عامل‌های 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 سیستم‌عامل‌های کامل، تنها برنامه‌ها و وابستگی‌های خود را ایزوله می‌کنند و همگی بر روی هسته سیستم‌عامل میزبان اشتراکی اجرا می‌شوند، که این امر باعث سبک‌تر، سریع‌تر و مقیاس‌پذیرتر شدن کانتینرها نسبت به ماشین‌های مجازی باشند. از این رو، کانتینرها برای محیط‌هایی که نیازمند راه‌اندازی

سریع و مدیریت منابع مانند میکروسرویس‌ها و برنامه‌های کاربردی مبتنی بر Cloud هستند ایده‌آل می‌باشند [۱۵]. در کنار مزایای فراوان کانتینرها، برخلاف ماشین‌های مجازی در امنیت و ایزولاسیون داده‌ها محدودیت‌هایی دارند و ممکن است نیازمند ابزارهای پیچیده‌تر برای مدیریت لاگ‌ها و نظارت باشند، که می‌تواند پیاده‌سازی و نگهداری آنها را چالش برانگیز سازد. تکنولوژی کانتینر ریشه در مفهوم چارچوب‌های Unix مانند chroot دارد که در دهه ۱۹۷۰ معرفی شد. اما، پیشرفت‌های اصلی در این زمینه با ظهور Docker در سال ۲۰۱۳ آغاز شد. داکر یک پلتفرم متن باز است که استانداردسازی 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 [۱۶]:

- موتورهای کانتینر<sup>۳۷</sup>

- هماهنگ سازی کانتینر<sup>۳۸</sup>

موتورهای کانتینری مانند Docker Engine و Containerd ابزارهایی هستند که 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. Namespaces بخشی از هسته لینوکس که امکان جداسازی عناصری مثل شبکه، فرایندها و فضای فایل سیستم را فراهم می‌کند. هر کانتینر در یک namespace جداگانه اجرا می‌شود که استقلال آن را نسبت به دیگر برنامه‌ها تضمین می‌کند. 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.

برای مدیریت و مقیاس‌بندی کانتینرها در محیط‌های تولید، ابزارهای هماهنگ سازی مانند Kubernetes و Docker Swarm کاربرد دارند. این ابزارها به توسعه‌دهندگان این امکان را می‌دهند که خوشه<sup>۳۹</sup> های بزرگ کانتینری را مدیریت کنند و برنامه‌ها را با انعطاف‌پذیری و دقت بالا مقیاس بندی نمایند. راجع به این موضوع در قسمت بعدی بیشتر صحبت خواهد شد.

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 لایه 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

<sup>۳۷</sup> Container Engine

<sup>۳۸</sup> Container Orchestration

<sup>۳۹</sup> Cluster



شکل ۲-۴: معماری لایه ای تصویر داکر

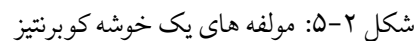
نوبه خود باعث کاهش حجم داده‌های مورد نیاز برای ذخیره‌سازی و انتقال می‌شود. زمانی که Dockerfile نوشته می‌شود، هر دستور (مانند RUN، COPY و FROM) یک لایه جدید در تصویر داکر ایجاد می‌کند. این لایه‌ها به ترتیبی که در داکر فایل آمده‌اند، روی هم اضافه می‌شوند. داکر از یک فایل سیستم Union استفاده می‌کند که به آن این اجازه را می‌دهد تا لایه‌های مختلف را به گونه‌ای ترکیب کند که به نظر یک فایل سیستم یکپارچه است [۱۷]. ساختار لایه ای تصویر داکر را در شکل ۲-۴ مشاهده می‌کنید.

## ۳-۲-۲ هماهنگ سازی کانتینرها (کوبرنتیز)

در دنیای توسعه نرم‌افزار، استفاده از معماری‌های مبتنی بر میکروسرویس‌ها و کانتینرها افزایش یافته است که هر دو نیازمند مدیریت دقیق و خودکار سرویس‌ها در محیط‌های تولید هستند. در محیط‌های پویا و با مقیاس بزرگ که دستگاه‌ها و خدمات به طور مکرر تغییر می‌کنند، تقریباً غیرممکن است که با نیروی کار دستی، سرویسی با دسترسی بالا ارائه داد. در چنین شرایطی، ابزارهای هماهنگ سازی نقش حیاتی ایفا می‌کنند. آنها به خودکارسازی مدیریت کانتینرها، مدیریت شبکه و نظارت بر سلامت سیستم کمک می‌کنند. بدون هماهنگ سازی تیم‌های توسعه و عملیات با چالش‌های عدیده‌ای از جمله کنترل ناموفق بر پیکربندی‌ها، مشکلات مربوط به برقراری ارتباط بین سرویس‌ها، دشواری‌های مربوط به مقیاس‌پذیری و برقراری تعادل بار مواجه می‌شوند. در میان ابزارهای هماهنگ سازی Kubernetes به عنوان یکی از پیشروان بازار شناخته می‌شود که امکان مدیریت خودکار مجموعه‌های بزرگی از کانتینرها را فراهم می‌آورد. کوبرنتیز یک پلتفرم هماهنگ‌سازی کانتینر است که فرایند زمان‌بندی<sup>۴۰</sup>، خودکارسازی استقرار، مدیریت و مقیاس‌گذاری اپلیکیشن‌های کانتینری را تسهیل می‌کند.

<sup>۴۰</sup>Scheduling





یک سیستم کوبرنتیز با تمام اجزای آن را یک خوشه<sup>۴۱</sup> می‌گویند. هر خوشه شامل یک یا چند گره<sup>۴۲</sup> است که می‌توانند فیزیکی<sup>۴۳</sup> یا مجازی باشند. این گره‌ها به دو دسته اصلی یا همان سطح کنترل<sup>۴۴</sup> و کارگر<sup>۴۵</sup> تقسیم می‌شوند. گره اصلی به عنوان مغز متفکر کوبرنتیز عمل می‌کند و وظایف مدیریتی خوشه را بر عهده دارد. این گره شامل مولفه‌های اصلی زیر است:

- **API Server:** نقطه اصلی دریافت فرمان‌های کوبرنتیز به صورت REST<sup>۴۶</sup> است و آن را پردازش می‌کند. این سرور مسئول اعتبارسنجی درخواست‌ها و اجرای آن‌ها بر روی خوشه است. همچنین، این مولفه به عنوان بخشی از مولفه‌های دیگر گره اصلی عمل می‌کند تا اطمینان حاصل شود که دستورات به درستی اجرا می‌شوند.
- **Scheduler:** مولفه‌ای است که تصمیم می‌گیرد کدام پادها بر روی کدام گره‌های کاری قرار گیرند. این فرایند بر اساس منابع موجود و الزامات مشخص شده برای پادها صورت می‌گیرد. علاوه براین، به طور مداوم وضعیت خوشه را رصد می‌کند تا بهترین تصمیم‌ها را برای مکان‌یابی پادها بگیرد.
- **Controller Managers:** مجموعه‌ای از فرآیندهایی است که حلقه‌های نظارتی را اجرا می‌کنند. این کنترل کننده‌ها وضعیت خوشه را با حالت مطلوب مطابقت می‌دهند. به عنوان مثال، اگر یک پاد از کار افتاده باشد، یک کنترل کننده وظیفه دارد تا یک پاد جدید را برای جایگزینی ایجاد کند.

- Cluster<sup>¶1</sup>
- Node<sup>¶2</sup>
- Bare-metal<sup>¶3</sup>
- Control plane<sup>¶4</sup>
- Worker<sup>¶5</sup>
- Representational State Transfer<sup>¶6</sup>

- 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 کانتینرهای درون آن را در خود ذخیره می‌کند.

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

- Kubelet: این مولفه وظیفه مدیریت سلامت پادها را بر عهده دارد. علاوه بر این اطمینان حاصل می‌کند که کانتینرها 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 برعهده دارند. این مولفه ارتباطات شبکه بین کانتینرها را تسهیل می‌کند و از قوانین IPTables برای مسیریابی ترافیک استفاده می‌کند.

در کوبرنتیز، چندین روش و نوع ذخیره‌سازی داده وجود دارد که می‌تواند بر اساس نیازهای برنامه و محیط اجرایی انتخاب شود. Volumes یکی از روش‌های اصلی برای ذخیره‌سازی داده‌ها در کوبرنتیز است. هر Volume به یک پاد متصل می‌شود و از طول عمر آن با پاد مرتبط است. انواع Volumes شامل:

Volumes یکی از روش‌های اصلی برای ذخیره‌سازی داده‌ها در کوبرنتیز است. هر Volume به یک Pod متصل می‌شود و از طول عمر آن Pod پیروی می‌کند. انواع Volumes شامل:

emptyDir: یک Volume خالی که با شروع Pod ایجاد می‌شود و با حذف Pod نیز حذف می‌شود. مناسب برای ذخیره‌سازی داده‌های موقت. hostPath: دسترسی به فایل‌ها و دایرکتوری‌های موجود در نود میزبان. این روش می‌تواند مشکلات امنیتی به همراه داشته باشد. nfs: استفاده از System File Network برای اشتراک‌گذاری فایل‌ها بین Podها. persistentVolumeClaim (PVC): یک نوع Volume که از Volume Persistent استفاده می‌کند و برای ذخیره‌سازی پایدار و دائمی مناسب است.

## فصل ۳

# MLOps

### ۱-۳ مقدمه

در حالی که مدل‌های یادگیری ماشین به طور گسترده 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 موثر 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 خطا به منظور 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 می‌کنیم.

### ۲-۳ تعریف مفاهیم اولیه

MLOps یا عملیات یادگیری ماشین به مجموعه‌ای از فرایندها، 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 DevOps است به گونه‌ای که این اطمینان حاصل شود که مدل‌ها به طور مؤثر توسعه، استقرار، پایش و به‌روزرسانی می‌شوند. هدف MLOps افزایش سرعت، قابلیت اطمینان و مقیاس‌پذیری مدل‌های یادگیری ماشین و فرایند توسعه این مدل‌ها در تولید است؛

درحالی که خطرات ناشی از ریسک عدم موفقیت را نیز کاهش می دهد. همچنین به کارگیری MLOps فرایند مدیریت را ساده تر کرده، کیفیت را افزایش می دهد و استقرار مدل های یادگیری عمیق و یادگیری ماشین در محیط های تولید با مقیاس بزرگ را خودکار می کند. لذا می توان گفت یکی از اهداف MLOps، بهبود خودکارسازی و ارتقای کیفیت مدل های تولید و درعین حال توجه به الزامات تجاری و نظارتی است.

استقرار مدل های یادگیری ماشین روی محیط عملیاتی در MLOps اهمیت زیادی دارد، زیرا به سازمان ها کمک می کند تا مطمئن شوند که مدل هایشان در طول زمان دقیق، قابل اعتماد و کارآمد هستند. به طور کلی، MLOps با خودکار کردن بسیاری از مراحل مربوط به استقرار و مدیریت مدل های یادگیری ماشین، به دانشمندان و مهندسان داده اجازه می دهد تا با همکاری یکدیگر به ارائه سریع تر و کارآمدتر مدل های یادگیری ماشین دست یابند.

## ۳-۲-۱ اصول

برای تسهیل در رسیدن به اهداف فوق، تیم های MLOps از اصول زیر استفاده می کنند:

۱. خط لوله خودکار CI/CD و هماهنگ سازی جریان کار<sup>۱</sup>: خودکارسازی CI/CD شامل مراحل ساخت، آزمایش، تحویل و استقرار است که به توسعه دهندگان نسبت به موفقیت یا شکست مراحل مختلف بازخورد سریعی را ارائه داده و بهره وری کلی را افزایش می دهد [۱۸]. در همین حال، هماهنگ سازی جریان کاری وظایف یک خط لوله یادگیری ماشین را با استفاده از گراف های بدون حلقه ی جهت دار<sup>۲</sup> هماهنگ می کند، که ترتیب اجرای وظایف را با توجه به روابط و وابستگی ها تعیین می کند. ترکیب این دو رویکرد می تواند به بهبود عملکرد و کارایی تیم های توسعه و داده کاوی کمک کند [۱۹، ۲۰].

۲. کنترل نسخه مدل های یادگیری ماشین، مجموعه داده ها و کد منبع: با استفاده از نسخه بندی مدل، داده و کد منبع، می توان هر تغییر و اصلاحی را در طول زمان دنبال کرد، که این امر به توسعه دهندگان و محققان اجازه می دهد تا به راحتی به نسخه های قبلی بازگردند و نتایج را بازبینی کنند. این قابلیت برای حفظ یکپارچگی و شفافیت در پروژه های نرم افزاری و علمی بسیار حیاتی است [۱۸].

۳. نظارت و آموزش مداوم مدل یادگیری ماشین: آموزش مداوم<sup>۳</sup> در یادگیری ماشین به معنای آموزش دوره ای مدل های یادگیری ماشین بر اساس داده های جدید است. این فرآیند همیشه شامل یک مرحله ارزیابی برای سنجش تغییرات

<sup>۱</sup> Workflow

<sup>۲</sup> Directed Acyclic Graph (DAG)

<sup>۳</sup> Continuous Training (CT)

کیفیت مدل است [۲۱]. نظارت مداوم به معنای ارزیابی دوره‌ای داده‌ها، مدل‌ها (مانند دقت پیش‌بینی)، کد منبع و منابع زیرساختی است تا خطاها یا تغییرات احتمالی که بر کیفیت محصول تاثیر می‌گذارند، شناسایی شوند. این فرآیند به توسعه‌دهندگان امکان می‌دهد تا به سرعت مشکلات را شناسایی و برطرف کنند و از افت عملکرد مدل جلوگیری کنند. یکی از دلایل لزوم آموزش مداوم، رانش داده یا مدل<sup>۴</sup> است، که به تغییرات تدریجی در داده‌ها یا عملکرد مدل در طول زمان اشاره دارد و می‌تواند باعث کاهش دقت پیش‌بینی‌ها شود [۲۲]. 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 توسعه‌دهندگان کمک می‌کند تا مدل‌ها را به‌روز نگه داشته و از تاثیرات منفی رانش داده یا مدل جلوگیری کنند [۲۳].

۴. ثبت فراداده<sup>۵</sup> 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 [۲۰]. 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 توسعه‌دهندگان کمک می‌کند تا تغییرات و نتایج را به دقت مستند کرده و در صورت نیاز به نسخه‌های قبلی بازگردند [۲۴].

۵. حلقه‌های بازخورد<sup>۶</sup>: حلقه‌های بازخورد به توسعه‌دهندگان اجازه می‌دهند تا به‌طور مداوم مدل‌ها را بهبود بخشند، 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 [۲۴]. 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<sup>۷</sup> است. این اصل بر امکان همکاری مشترک افراد بر روی داده‌ها، مدل‌ها و کدها تاکید دارد. علاوه بر جنبه‌های فنی، اصل همکاری به ایجاد فرهنگ کاری مشارکتی توجه دارد که هدف آن کاهش ایزوله سازی‌های حوزه‌ای بین نقش‌های مختلف است. چنین رویکردی باعث می‌شود تا افراد با تخصص‌های گوناگون به‌طور هم‌افزا با یکدیگر کار کنند، دانش خود را به اشتراک بگذارند و از هم بیاموزند.

---

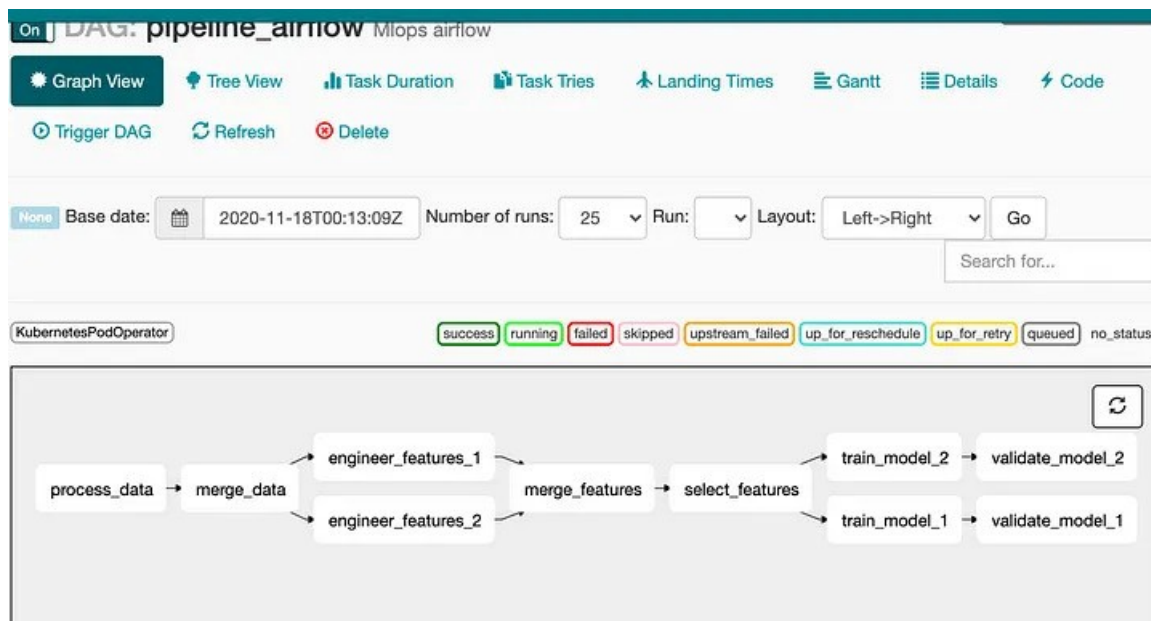
<sup>۴</sup>Data or Model Drift

<sup>۵</sup>Metadata

<sup>۶</sup>feedback loops

<sup>۷</sup>Collaboration

Workflow Orchestration<sup>^</sup>  
Extract, Transform, Load (ETL)<sup>q</sup>  
Feature Store<sup>o</sup>

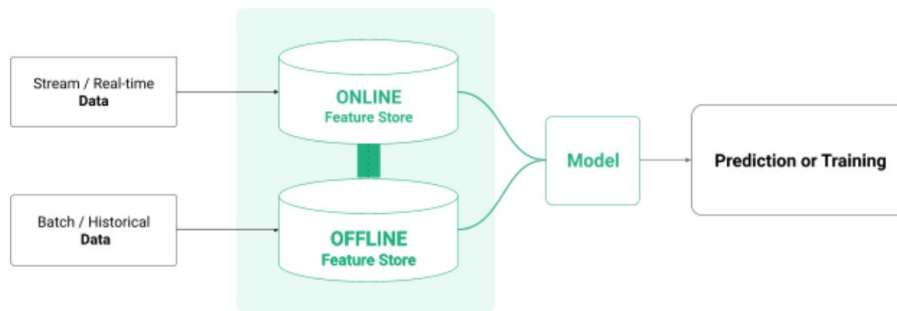


شکل ۳-۱: خط لوله در Apache Airflow

که نیاز به پردازش حجم زیادی از داده‌ها در مدت زمان طولانی تر 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 بلادرنگ 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 متمرکز ذخیره کرده و به راحتی در پروژه‌های مختلف به اشتراک بگذارند، که این امر به تسریع 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 [۲۳]. از ابزار معروف متن باز برای می‌توان به Feast [۲۸] اشاره نمود.



شکل ۳-۲: انبار ویژگی

### بانک مدل

بانک مدل<sup>۱۱</sup> 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. هم چنین 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. هم چنین به هنگام بروز مشکل در مدل‌های جدید، می‌توان از مدل‌های قابل قبول قبلی برای محیط عملیاتی استفاده کرد [۲۹]. از ابزار معرفی متن باز می‌توان به MLflow [۳۰] اشاره کرد.

### انبار فراداده

انبار فراداده یادگیری ماشین<sup>۱۲</sup> 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. یکی از کاربردهای اصلی انبار فراداده‌ها، مدیریت کارآمد پروژه‌های پیچیده یادگیری ماشین است [۲۴]. 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.

<sup>۱۱</sup> Model Registry

<sup>۱۲</sup> ML Metadata Store



## استقرار مدل

استقرارکردن مدل<sup>۱۳</sup> 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 بپردازند. 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<sup>۱۴</sup> برای پردازش حجم بالای داده‌ها پیاده‌سازی شود. در محیط‌های عملیاتی، فرآیند استقرار مدل به سه شکل اصلی بلادرنگ، دسته‌ای و بدون سرور پیاده‌سازی می‌شود [۲۹].

در استنتاج بلادرنگ<sup>۱۵</sup>، 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 حسگرها و 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. تکنولوژی‌های مانند RESTful APIs و gRPC معمولاً برای پیاده‌سازی این نوع سرویس‌دهی استفاده می‌شوند.

استنتاج دسته‌ای<sup>۱۶</sup> 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 [۳۱] و Hadoop MapReduce معمولاً برای پیاده‌سازی استنتاج دسته‌ای استفاده می‌شوند.

در استنتاج بدون سرور<sup>۱۷</sup>، 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 مقرون‌به‌صرفه 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 متن باز برای استقرار مدل می‌توان به Knative [۳۲] اشاره کرد.

<sup>۱۳</sup> Model Serving

<sup>۱۴</sup> Batch

<sup>۱۵</sup> Real-time Inference

<sup>۱۶</sup> Batch Inference

<sup>۱۷</sup> Serverless Inference

## نظارت

نظارت<sup>۱۸</sup> در یادگیری ماشین یکی از مولفه‌های حیاتی برای تضمین عملکرد بهینه مدل‌ها و زیرساخت‌های مرتبط است. نظارت مداوم بر مدل‌های یادگیری ماشین به دلایلی از جمله اطمینان از دقت پیش‌بینی‌ها، شناسایی ناهنجاری‌ها و بهبود مداوم عملکرد مدل‌ها ضروری است [۳۳]. ابزارهایی مانند TensorBoard، Kubeflow و MLflow نیز نقش مهمی در نظارت بر مدل‌های یادگیری ماشین ایفا می‌کنند. TensorBoard به ویژه برای مصورسازی و تحلیل مراحل مختلف آموزش مدل‌ها مفید است.

نظارت در یادگیری ماشین تنها به مدل‌ها محدود نمی‌شود؛ بلکه زیرساخت‌های مرتبط با یادگیری ماشین نیز نیاز به نظارت دارند. این نظارت شامل نظارت بر فرآیندهای CI/CD، هماهنگی سرویس‌ها، خوشه‌های عملیاتی کوبرنتیز و گره‌های محاسباتی می‌شود [۲۴]. یکی از ابزارهای رایج برای نظارت، Prometheus است که به همراه Grafana برای مصورسازی داده‌ها استفاده می‌شود. علاوه بر این پشته ELK (Elasticsearch، Logstash، Kibana) نیز یک مجموعه قدرتمند برای جستجو، تحلیل و مصورسازی لاگ‌های سیستم است که می‌تواند به شناسایی و رفع سریع مشکلات کمک کند. ابزارهای نظارتی به مهندسان اجازه می‌دهند تا هر گونه 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 واحد پردازش گرافیکی<sup>۱۹</sup> است که برای پردازش داده‌ها و اجرای الگوریتم‌های پیچیده می‌باشد. زیرساخت‌ها می‌توانند به دو شکل توزیع شده<sup>۲۰</sup> 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 ابزار محبوب برای مدیریت و سازآرایی محاسبات توزیع شده، کوبرنتیز است که امکان مدیریت کانتینرها و توزیع بار کاری بین گره‌ها را فراهم می‌کند. هم چنین، Red Hat OpenShift نیز به عنوان یک پلتفرم دیگر شناخته می‌شود که قابلیت‌های مشابهی ارائه می‌دهد [۲۹].

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

<sup>۱۸</sup>Monitoring

<sup>۱۹</sup>GPU

<sup>۲۰</sup>Distributed

بهینه‌سازی شده‌اند، استفاده می‌شوند. در دستگاه‌های لبه<sup>۲۱</sup> به دلیل محدودیت‌های فضا، توان محاسباتی و مصرف انرژی، اجرای مدل‌های یادگیری عمیق پیچیده به 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 کم دقت، می‌توان محاسبات را سریع‌تر و با مصرف انرژی کمتری انجام داد. تکنیک دیگر، هرس کردن<sup>۲۲</sup> است که شامل حذف اتصالات غیرضروری و وزن‌های کوچک در شبکه عصبی می‌شود. این فرآیند باعث کاهش تعداد پارامترهای مدل می‌شود، بدون آنکه تاثیر 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

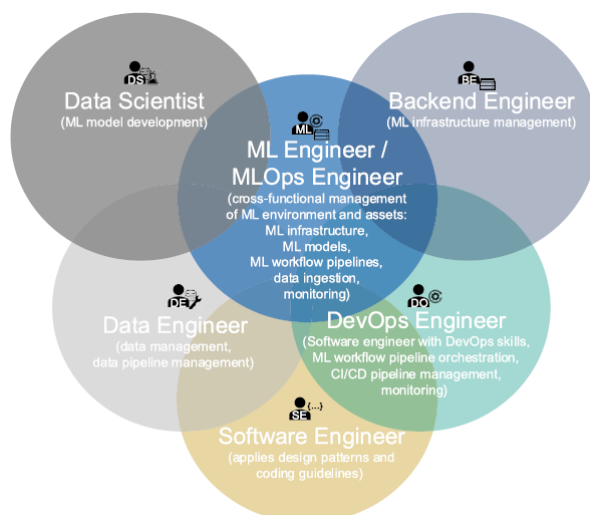
مخزن کد منبع به عنوان یک نقطه مشترک برای نگهداری و مدیریت کدهای مربوط به مدل‌های یادگیری ماشین یک سازمان عمل می‌کند. با استفاده از سیستم‌های مدیریت نسخه مانند گیت، تیم‌ها می‌توانند به راحتی تغییرات کد را پیگیری کرده و در صورت لزوم به نسخه‌های قبلی کد بازگردند. این مخزن همچنین به خودکارسازی فرآیند CI/CD 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. می‌توان از ابزار متن باز برای پیاده‌سازی آن به GitLab [۳۴] و Gerrit [۳۵] 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 به‌روزرسانی 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 هم چنین داده‌ها استفاده می‌شوند [۲۴]. از ابزارهای مناسب برای این کار می‌توان به Jenkins [۳۶] و GitLab CI [۳۴] نام برد.

<sup>۲۱</sup> Edge Devices

<sup>۲۲</sup> Pruning



شکل ۳-۳: نقش ها و اشتراکات آنها در پارادایم MLOps

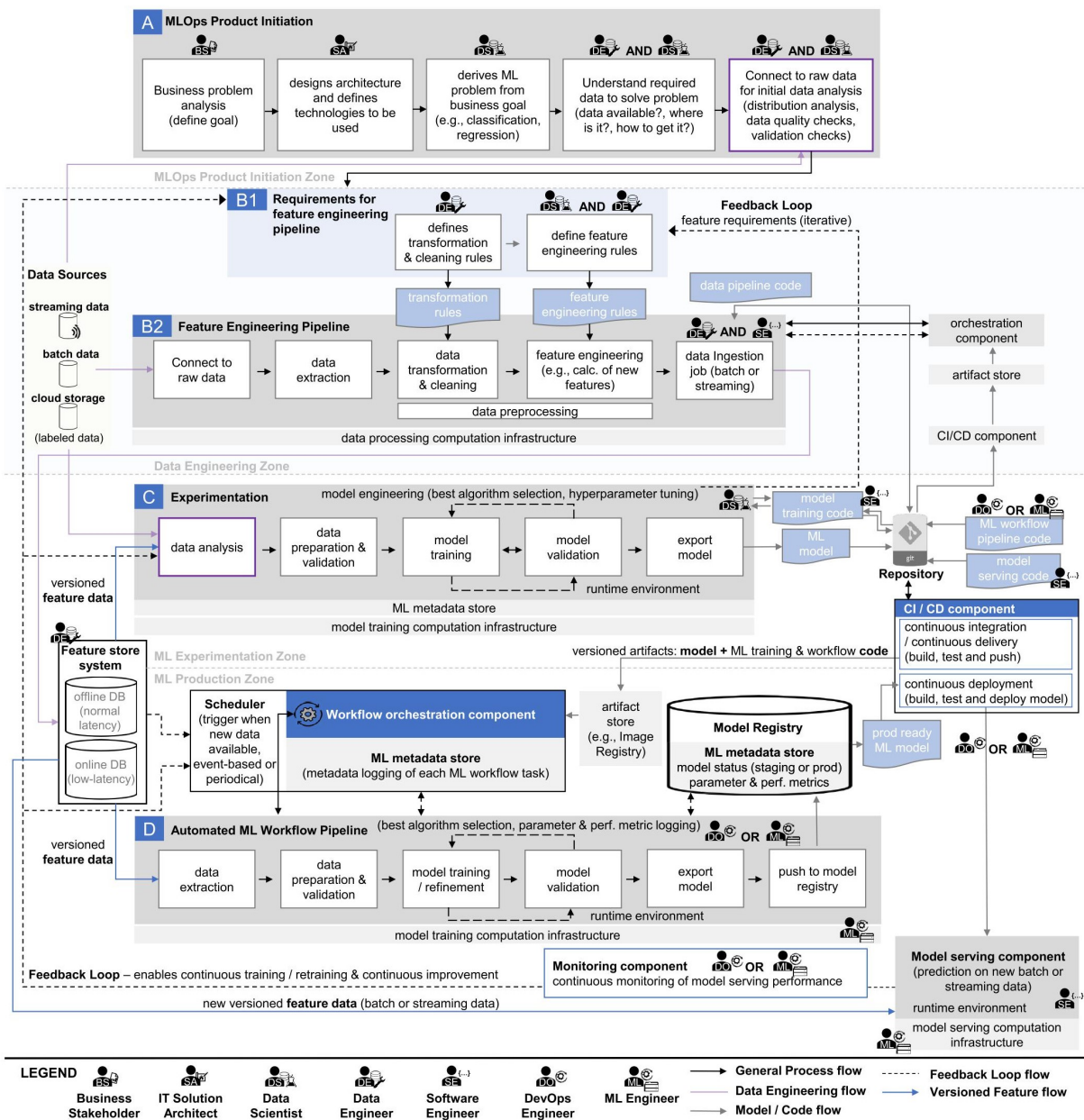
### ۳-۲-۳ نقش ها

در تولید یک پلتفرم MLOps، نقش های متعددی وجود دارد که همکاری آن ها برای طراحی، مدیریت، اتوماسیون و بهره برداری از سیستم های یادگیری ماشین در محیط تولید بسیار حیاتی است. در ابتدا سهام دار کسب و کار<sup>۲۳</sup> وظیفه تعیین اهداف کسب و کار و استراتژی بازگشت سرمایه محصول یادگیری ماشین را بر عهده دارد. معمار، معماری سیستم را طراحی کرده و فناوری های مناسب را انتخاب می کند. دانشمند داده مسئله کسب و کار را به مسئله یادگیری ماشین ترجمه کرده و مدل ها را مهندسی می کند. مهندس داده خط لوله های داده و ویژگی را ایجاد و مدیریت می کند و داده ها را به درستی به سیستم های پایگاه داده و انبار ویژگی ها تزریق می کند. مهندس نرم افزار با استفاده از الگوهای طراحی، مسئله یادگیری ماشین را به یک محصول مهندسی شده تبدیل می کند. مهندس DevOps خودکارسازی CI/CD، ساز آرای جریانی یادگیری ماشین و استقرار مدل در تولید را تضمین می کند. در نهایت، مهندس MLOps نقش ترکیبی از مهارت های چندگانه را دارد و زیرساخت یادگیری ماشین را ایجاد و مدیریت کرده، خط لوله های جریان کاری را خودکار می کند و مدل ها و زیرساخت را در تولید نظارت می کند. این نقش ها که در شکل ۳-۳ نشان داده شده است، با همکاری و هماهنگی نزدیک می توانند MLOps را به شکلی مؤثر و کارآمد پیاده سازی کنند، که نتیجه آن یک سیستم یادگیری ماشین پایدار و قابل اعتماد در محیط تولید خواهد بود. همکاری میان این نقش ها تضمین می کند که تمام جنبه های مربوط به توسعه، استقرار و نگهداری مدل های یادگیری ماشین به درستی مدیریت شود و به اهداف کسب و کار دست یابند.

<sup>۲۳</sup> Business Stakeholder

### 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 ۳-۴ 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 متن‌باز، استفاده از سرویس‌های ابری یا



شکل ۳-۴: معماری جامع MLOps

رویکردهای ترکیبی پیاده‌سازی کنند. نرم‌افزارهای سازمانی و خدمات ابری اغلب از طریق API ها با ابزارهای متن‌باز یکپارچه می‌شوند و امکان ترکیب بی‌دردسر فناوری‌های مختلف را فراهم می‌کنند. این معماری یک معماری جامع بوده و هر سازمان می‌تواند برحسب نیاز و هدف خود در حل مسئله یادگیری ماشین، این معماری را شخصی سازی کرده و از پیاده سازی قسمت هایی از آن صرف نظر کند.

### 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 وارد فرآیند می‌شود. 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 پروژه است. نکته مهم در این فرآیند، نیاز به داده‌های برجسته‌گذاری شده است که برای الگوریتم‌های نظارت‌شده ضروری هستند. در این معماری منابع داده از قبل دارای داده‌های برجسته‌گذاری شده بوده‌اند، زیرا فرآیند برجسته‌گذاری در مراحل قبلی انجام شده است.

### خط لوله مهندسی ویژگی

در فرآیند توسعه مدل‌های یادگیری ماشین، مهندسی ویژگی‌ها<sup>۲۴</sup> به عنوان یکی از گام‌های حیاتی شناخته می‌شود که مستلزم تعیین نیازمندی‌های اساسی و پیاده سازی 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<sup>۲۵</sup> مانند نرمال‌سازی و تجمیع، و همچنین قواعد پاک‌سازی داده‌ها<sup>۲۶</sup> 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

<sup>۲۴</sup> Feature Engineering

<sup>۲۵</sup> Data Transformation

<sup>۲۶</sup> Data Cleaning

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، به‌روزرسانی 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<sup>۲۷</sup>، داده‌های دسته‌ای<sup>۲۸</sup> یا داده‌های ذخیره شده در ابر<sup>۲۹</sup> باشند. داده‌های جریانی به صورت پیوسته و بلادرنگ دریافت می‌شوند، داده‌های دسته‌ای ثابت به صورت دوره‌ای و در حجم بالا جمع‌آوری و پردازش می‌شوند و داده‌های ذخیره شده در ابر از طریق سیستم‌های ابری مقیاس پذیر و انعطاف پذیر ذخیره می‌شوند. اتصال به منابع داده باید به گونه‌ای 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.
- پیش پردازش داده‌ها: در این مرحله، داده‌های استخراج شده برای تبدیل به فرم قابل استفاده، پیش پردازش می‌شوند. پیش پردازش شامل مراحل مختلفی مانند پاکسازی داده‌ها، مدیریت مقادیر مفقود، حذف نویز، و نرمال‌سازی مقادیر است. هدف اصلی این مرحله، آماده‌سازی داده‌ها به گونه‌ای است که بتوانند به عنوان ورودی‌های مدل یادگیری ماشین استفاده شوند.
- استخراج ویژگی‌های جدید و پیشرفته: یکی از مهم‌ترین مراحل در خط لوله تولید ویژگی‌ها، استخراج ویژگی‌های جدید و پیشرفته 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 به گونه‌ای انجام شود که دسترسی سریع و کارآمد به داده‌ها برای مراحل بعدی آموزش مدل فراهم شود.

<sup>۲۷</sup> Streaming Data

<sup>۲۸</sup> batch data

<sup>۲۹</sup> Cloud Storage

در خط تولید ویژگی‌ها، مهندس نرم افزار به کمک مهندس داده کدهای مورد نیاز برای CI/CD و سازآرایی را تعریف می‌کند تا وظایف خط تولید ویژگی‌ها به درستی هماهنگ شوند. این نقش شامل تنظیم منابع زیرساختی برای اطمینان از مقیاس پذیری 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 استفاده می‌شود. برخی از این ابزارها شامل Apache Spark، Apache Kafka و ابزارهای ETL سنتی مانند Apache Airflow هستند. Apache Spark به دلیل توانایی بالای آن در پردازش موازی و تحلیل داده‌های بزرگ بسیار محبوب است. به عنوان مثال، در یک پروژه پردازش زبان طبیعی<sup>۳۰</sup> با استفاده از اسپارک، داده‌های متنی بزرگ پردازش و ویژگی‌های متنی جدید محاسبه شدند [۳۷]. در پروژه دیگری در یک موسسه مالی، داده‌های اعتباری مشتریان با استفاده از اسپارک پردازش و ویژگی‌های مرتبط برای مدل ریسک اعتباری ایجاد شدند [۳۸]. Apache Kafka [۳۹] نیز برای بارگذاری داده‌های جریانی بلادرنگ به کار گرفته می‌شود. هم چنین 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 نرم افزار انجام می‌شود. قبل از شروع به کار، مهندس داده به همراه مهندس نرم افزار برای اطمینان از عملکرد درست ابزار و منابع، محیط و سخت افزار 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، پاک سازی 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 موثری آموزش داده شوند.

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

<sup>۳۰</sup> Natural Language Processing



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 مستندات 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 تحویل مدل را آغاز می‌کند. در مرحله ساخت، مصنوعات مدل<sup>۳۱</sup> و کدهای مرتبط ایجاد می‌شوند. در مرحله آزمون، صحت و عملکرد مدل بررسی می‌شود و در نهایت، در مرحله تحویل مدل نهایی به مخزن مصنوعات ارسال می‌شود تا برای استفاده در محیط عملیاتی آماده باشد.

در مرحله آزمایش، ابزارهای مبتنی بر Notebook مانند 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 نظیر کوبرنتیز استفاده می‌کنند. در این سیستم، یک مولفه ارکستراسیون وظایف مختلف را در جریان کاری خودکار یادگیری ماشین هماهنگ می‌کند. این مولفه وظایف را به محیط‌های مجزا (مانند کانتینرها) تخصیص داده و فراداده‌های هر وظیفه را در قالب لاگ‌ها، و سایر اطلاعات جمع‌آوری می‌کند. مراحل اجرای این فرآیند که به قسمت قبل خیلی شباهت دارد به صورت زیر است:

<sup>۳۱</sup> Model Artifacts

- استخراج داده‌ها: اولین مرحله در این فرآیند، استخراج داده‌ها از سیستم‌های انبار و 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 شده 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، فراخوانی می‌شوند. خط لوله CD وظیفه ساخت و آزمایش مدل و کدهای استقرار مدل برای استقرار در محیط عملیاتی را بر عهده دارد. مؤلفه استقرار مدل مانند Knative پیش‌بینی‌ها را بر اساس داده‌های جدید و دیده نشده از سیستم انبار و ویژگی‌ها انجام می‌دهد. این مؤلفه می‌تواند توسط مهندس نرم‌افزار به صورت آنلاین برای پیش‌بینی‌های زمان واقعی یا دسته‌ای برای داده‌های کلان طراحی شود. برای پیش

بینی‌های زمان واقعی، ویژگی‌ها باید از پایگاه داده آنلاین با تأخیر کم دریافت شوند، در حالی که برای پیش‌بینی‌های دسته‌ای، ویژگی‌ها می‌توانند از پایگاه داده آفلاین با تأخیر معمولی دریافت شوند. 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 A/B 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 کاهش دقت پیش‌بینی‌ها تشخیص داده شود، اطلاعات از طریق حلقه بازخورد ارسال می‌شود. این حلقه امکان آموزش و بازآموزی مداوم<sup>۳۲</sup> 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 ویژگی‌های 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 دهند. رانش داده به دو نوع اصلی تقسیم می‌شود:

**رانش مفهوم<sup>۳۳</sup>:** تغییر در توزیع برجسب‌ها یا خروجی‌ها که نشان دهنده تغییر در الگوهای زیرین داده‌هاست.

**رانش ویژگی<sup>۳۴</sup>:** 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 مجدداً آموزش 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 Apache Airflow، Kubeflow، Pipelines و AWS SageMaker هستند. یک مثال از کاربرد صنعتی یک خط لوله خودکار یادگیری ماشین با استفاده از Airflow 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

<sup>۳۲</sup>Continuous Training (CT)

<sup>۳۳</sup>Concept Drift

<sup>۳۴</sup>Feature Drift

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 به‌عنوان 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 ۴

# طراحی یک پلتفرم MLOps

### ۱-۴ مقدمه

در این بخش می خواهیم مقدمه ای از طراحی و هدف آن بنویسیم

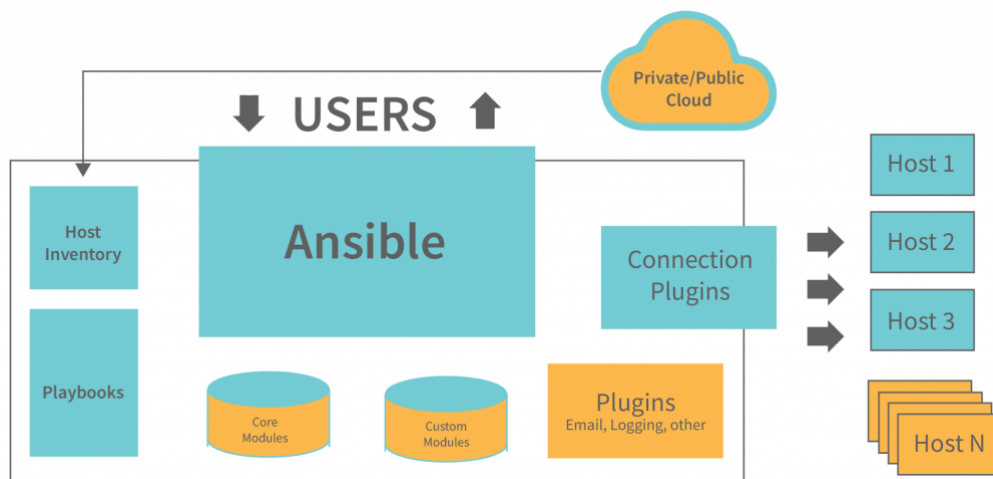
### ۲-۴ سیستم مدیریت پلتفرم

#### ۱-۲-۴ مدیریت پیکربندی و فراهم سازی زیرساخت

در دنیای فناوری اطلاعات، زیرساخت های تغییرناپذیر<sup>۱</sup> و تغییرپذیر<sup>۲</sup> دو رویکرد مهم در مدیریت و نگهداری سیستم ها هستند. زیرساخت های تغییرناپذیر به سیستم هایی اشاره دارند که پس از ایجاد، بدون تغییر باقی می ماندند و در صورت نیاز به تغییر، سیستم های جدید جایگزین آنها می شوند. این رویکرد با مزایایی همچون کاهش پیچیدگی های مدیریتی، افزایش قابلیت پیش بینی و کاهش ریسک های مرتبط با تغییرات ناخواسته همراه است [۹]. به کمک ابزارهایی مانند داکر و کوبرنتیز، پیاده سازی زیرساخت های تغییرناپذیر امکان پذیر است و از قابلیت مقیاس پذیری بالایی برخوردارند. سیستم های ابری غالباً از روش تغییرناپذیر استفاده کرده تا از مزایای آن بهره مند شوند. در مقابل، زیرساخت های تغییرپذیر به سیستم هایی اشاره دارند که می توانند به طور پویا تغییر کنند و تنظیمات و پیکربندی های جدید را بپذیرند. این رویکرد، انعطاف پذیری بیشتری را فراهم می کند و برای محیط هایی که نیاز به تغییرات مکرر دارند، مناسب تر است. با این حال، مدیریت تغییرات در زیرساخت های تغییرپذیر ممکن است چالش های بیشتری از جمله افزایش ریسک خطاها و نیاز به نظارت مداوم به همراه داشته باشد. انتخاب بین این دو رویکرد به نیازها و اولویت های سازمان بستگی دارد. در حالی که زیرساخت های تغییرناپذیر برای محیط های تولید با

---

Immutable Infrastructure<sup>۱</sup>  
Mutable Infrastructure<sup>۲</sup>



شکل ۴-۱: معماری Ansible

نیاز به ثبات و قابلیت پیش‌بینی بالا مناسب‌ترند، زیرساخت‌های تغییرپذیر برای محیط‌های توسعه و آزمایش که نیاز به انعطاف‌پذیری دارند، کاربرد بیشتری دارند [۸]. این دو مفهوم به صورت مستقیم با مدیریت پیکربندی و فراهم سازی 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<sup>۳</sup> 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.

<sup>۳</sup>Agent

## فراهم سازی زیرساخت

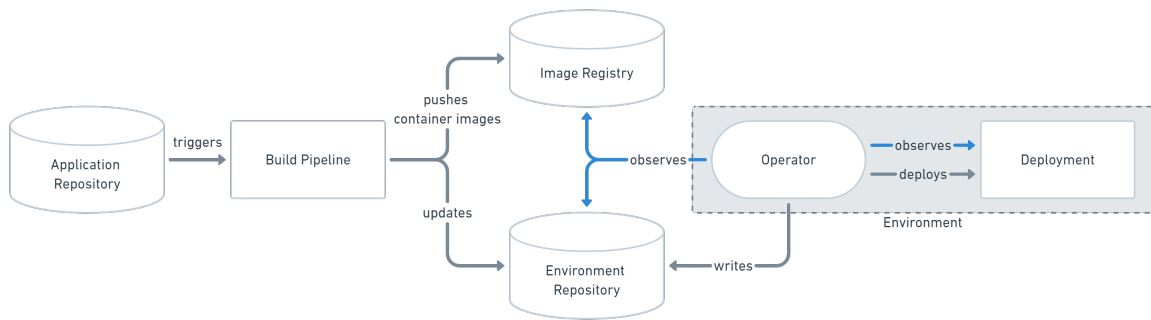
فراهم سازی زیرساخت فرآیندی است که به راه اندازی و پیکربندی اولیه زیرساخت های IT اختصاص دارد. این فرآیند شامل ایجاد و مدیریت منابعی مانند سرورها، پایگاه داده ها، شبکه ها و سایر اجزای زیرساختی است. فراهم سازی زیرساخت به صورت سنتی فرآیندی دستی و زمان بر بود، اما با ظهور ابزارهای نوین این فرآیند به شدت خودکار و ساده شده است. ابزارهای متن بازی مانند Terraform برای این کار استفاده می شوند. این ابزارها به کاربران امکان می دهند تا زیرساخت های خود را به صورت کد<sup>۴</sup> تعریف کنند [۸]. این رویکرد به ایجاد و مدیریت منابع زیرساختی به صورت قابل تکرار و پایدار کمک می کند. Terraform، یکی از پرکاربردترین ابزارهای فراهم سازی زیرساخت، از زبان HCL برای تعریف زیرساخت ها استفاده می کند. یکی از ویژگی های برجسته Terraform مدیریت وابستگی ها بین منابع است که امکان بازگردانی<sup>۵</sup> به وضعیت های قبلی را نیز فراهم می کند [۴۳]. از این ابزار ها غالباً برای زیرساخت ها تغییرناپذیر استفاده می شود.

در زیرساخت طراحی شده که از OpenStack برای ساخت و مدیریت ماشین های مجازی استفاده می شود، استفاده از Ansible به عنوان ابزار مدیریت پیکربندی به دلایل متعددی مناسب است. اولاً، Ansible با استفاده از Playbook های YAML امکان خودکارسازی مراحل پیکربندی را فراهم می کند، از جمله نصب نرم افزارهای مورد نیاز، تنظیمات شبکه و پیکربندی سرویس ها. این ویژگی باعث می شود که پیکربندی ها به صورت دقیق و بدون خطا انجام شود. ثانیاً، Ansible بدون نیاز به نصب عامل بر روی ماشین های مجازی کار می کند و از پروتکل SSH برای ارتباط استفاده می کند که این امر فرآیند پیکربندی را ساده تر و سریع تر می سازد. همچنین، تمامی مراحل پیکربندی به صورت کد تعریف می شوند که امکان اجرای مجدد و دقیق همان تنظیمات را بر روی ماشین های جدید فراهم می کند. به علاوه، Ansible دارای ماژول های متعددی برای تعامل با OpenStack است که می تواند فرآیند ایجاد و مدیریت ماشین های مجازی را بهینه تر کند. پس از پیکربندی اولیه VM ها، Ansible می تواند خوشه کوبرنتیز را به صورت خودکار راه اندازی و پیکربندی کند. این شامل نصب ابزارهای مورد نیاز، تنظیمات شبکه و پیکربندی سرویس های کوبرنتیز است.

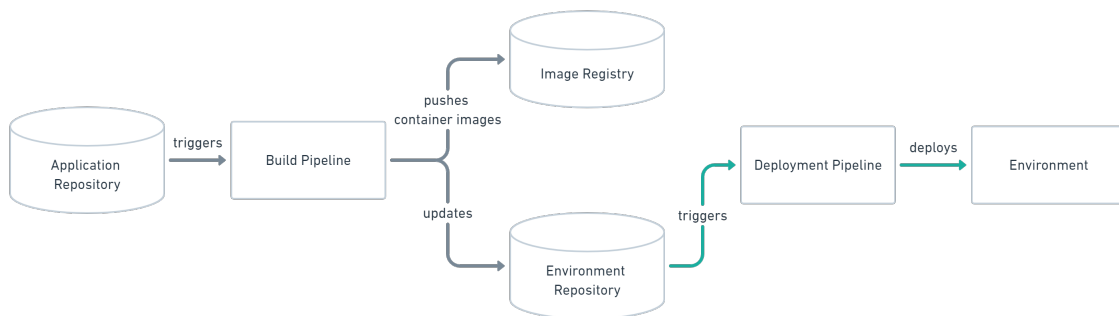
## ۴-۲-۲ خط لوله CI/CD

با افزایش استفاده از سیستم های ابری و رویکرد تغییرناپذیر، مفهومی با عنوان GitOps معرفی شد که از گیت برای مدیریت زیرساخت و پیکربندی بهره می برد. GitOps توسط Weaveworks در سال ۲۰۱۷ معرفی شد که رویکردی مدرن برای پیاده سازی خط لوله CD بر روی سیستم های ابری است. در حالی که ابزارهای سنتی تحویل مداوم عمدتاً از مدل Push استفاده

<sup>۴</sup> Infrastructure as Code  
<sup>۵</sup> Rollback



شکل ۴-۲: استقرار مبتنی بر Pull



شکل ۴-۳: استقرار مبتنی بر Push

می‌کنند، GitOps مدل Pull را معرفی می‌کند که به خصوص با کانتینرها و پیکربندی‌های اعلامی به خوبی کار می‌کند و آن را به یک روند محبوب در اکوسیستم بومی ابر تبدیل کرده است [۱۲]. از معروف ترین ابزار کلیدی که فرآیندهای GitOps را تسهیل می‌کند می‌توان به ArgoCD اشاره کرد.

همان طور که گفته شد دو رویکرد در پیاده سازی خط لوله CD وجود دارد. در مدل Pull (شکل ۴-۲) 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 می‌کنند. این مدل امنیت را افزایش می‌دهد زیرا نیازی به اعتبارنامه‌های دسترسی مستقیم برای توسعه‌دهندگان نیست. هم چنین 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 ۴-۳)، 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 [۱۲].



به منظور پیاده‌سازی یک ابزار متن‌باز برای مدیریت خط لوله های CI/CD که برای سیستم‌های ابری نیز مناسب باشد، رویکرد تغییرپذیر به همراه استراتژی استقرار مبتنی بر Push انتخاب شده است. انتخاب رویکرد تغییرپذیر به دلیل نیاز به 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. هم چنین روش Push نیز به دلیل سادگی و کارایی در اعمال به‌روزرسانی‌ها انتخاب شده است. با استفاده از این روش، می‌توانیم به‌روزرسانی‌ها را مستقیماً به سرورها ارسال کنیم و اطمینان حاصل کنیم که تمام سیستم‌ها به سرعت و بدون نیاز به مداخله دستی به‌روز 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.

## طراحی خط لوله

؟؟؟؟؟؟؟؟؟؟

## ۴-۲-۳ مخزن کد منبع

مخزن کد منبع<sup>۶</sup> 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 یکپارچه می‌شود. از آنجایی که یکی از شرط‌های پیاده‌سازی پلتفرم، متن‌باز بودن ابزارهای آن می‌باشد، GitLab برای

<sup>۶</sup>Source Code Repository

مدیریت مخزن کد منبع استفاده شده است. در این طراحی Jenkins به GitLab به عنوان مخزن کد متصل شده و هر تغییر در کد منبع باعث اجرا شدن یک خط لوله CI/CD مشخص توسط Jenkins می گردد.

#### ۴-۲-۴ مخزن مؤلفه ها

یک مخزن مؤلفه<sup>۷</sup> یک سیستم متمرکز برای ذخیره سازی، مدیریت و انتشار مؤلفه های نرم افزاری است. این مؤلفه ها شامل هر نوع فایل باینری، کتابخانه، ماژول، پکیج، پلاگین یا حتی مستنداتی می شود که در طول چرخه عمر توسعه نرم افزار تولید می شود. هدف اصلی این مخازن این است که به تیم های توسعه اجازه دهد تا به راحتی نسخه های مختلفی از مؤلفه ها را مدیریت و به اشتراک بگذارند، فرآیندهای ساخت و انتشار را ساده کنند و وابستگی ها را طور موثرتری مدیریت کنند. همچنین، از انتشار مؤلفه هایی که هنوز تست نشده اند یا از نظر امنیتی مشکلاتی دارند جلوگیری می کنند. یکی از ابزارهای محبوب و متن باز برای مدیریت مخازن مؤلفه ها، Nexus است. Nexus از فرمت های مختلف مؤلفه ها مانند Helm، apt، PyPI و Docker پشتیبانی می کند، که این امر آن را به یک ابزار چند منظوره برای انواع پروژه های نرم افزاری تبدیل می کند. مخازن مؤلفه مورد نیاز برای پیاده سازی پلتفرم در ۴ نوع raw، apt، PyPI و Docker می باشد.

#### مخازن APT

یک سیستم مدیریت بسته در سیستم عامل های مبتنی بر دبیان است که به کاربران اجازه می دهد تا بسته های نرم افزاری را به راحتی نصب، به روز رسانی و حذف کنند. از آنجایی بسته های استفاده شده در سرور ها غالباً یکسان می باشد، به منظور افزایش سرعت پیاده سازی و اعمال تغییرات و پیکربندی تمامی بسته های مورد استفاده و نصب نشده در محیط، در Nexus ذخیره خواهند شد. این امر با استفاده از مخازنی از نوع Proxy انجام خواهد شد. در پیاده سازی سیستم علاوه بر بسته های موجود در APT رسمی Ubuntu، از مخازن Containerd و Kubernetes نیز برای نصب و پیاده سازی کوپرنیتیز 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 بسته های نرم افزاری پایتون است که به توسعه دهندگان اجازه می دهد تا کتابخانه ها و ابزارهای خود را منتشر، به روز رسانی و مدیریت کنند. کاربران می توانند این بسته ها را به راحتی با استفاده از ابزار pip نصب کنند. همانند APT،

<sup>۷</sup>Artifact Repository

## مخزن Docker

مخزن raw

شکل از نکسوس بگذار

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 خوشه کوبرنتیز مستقر می‌شوند. استفاده از کوبرنتیز در پلتفرم‌های MLOps به دلیل قابلیت‌های منحصر به فرد آن در مدیریت خودکار، مقیاس‌پذیری و کارایی 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 به‌روزرسانی 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. این ویژگی‌ها باعث می‌شوند تا سازمان‌ها بتوانند مدل‌های خود را به طور مداوم بهبود داده و به صورت موثر در محیط‌های تولیدی مستقر کنند. 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 خوشه کوبرنتیز پیاده‌سازی می‌شوند را طراحی کرده و برای هر بخش یک ابزار مناسب معرفی می‌کنیم.

### ۴-۳-۱ مدیریت داده

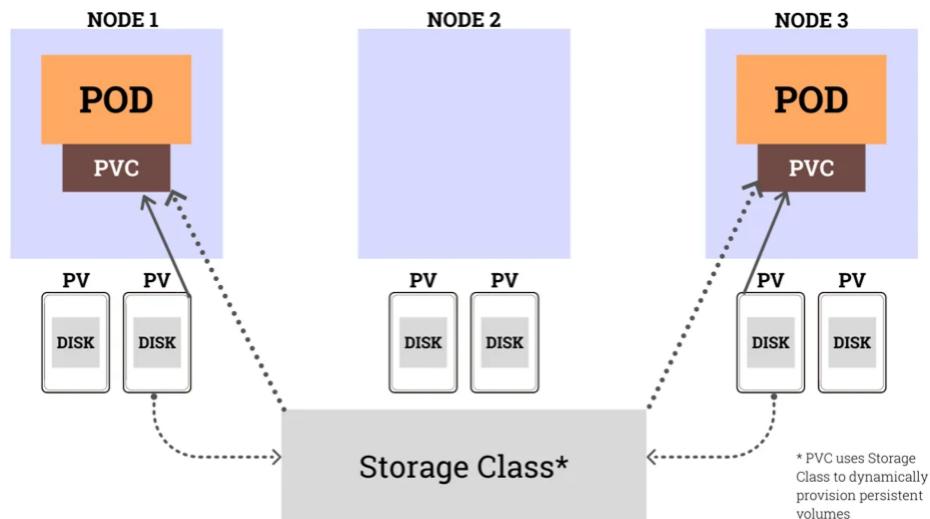
داده‌ها در پلتفرم MLOps نقش مهمی دارند. دانشمندان داده به منظور پیاده‌سازی یک مدل یادگیری ماشین نیاز به آموزش این مدل‌ها با استفاده از داده‌های از پیش آماده دارند. این داده‌ها می‌تواند به صورت متن، تصویر یا صوت باشد. لذا وجود یک محل ذخیره‌سازی داده برای نگه‌داری داده‌ها در این پلتفرم اساسی است. در کوبرنتیز، 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 خوشه 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 خوشه کوبرنتیز ساده‌تر و کارآمدتر شود. PVها می‌توانند از انواع مختلف ذخیره‌سازی مانند دیسک‌های محلی، شبکه‌های ذخیره‌سازی<sup>۸</sup> یا راه‌حل‌های 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<sup>۹</sup> است که به طور خودکار PVها را براساس نیازهای PVCها و کلاس‌های ذخیره‌سازی<sup>۱۰</sup> ایجاد می‌کند. این ویژگی، نیاز به ایجاد و مدیریت دستی PV را توسط مدیران خوشه 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 این ابزار ما PVها مورد نیاز برای ذخیره‌سازی داده‌های پایگاه داده و ذخیره‌سازی شیء مانند MinIO را ایجاد می‌کنیم.

<sup>۸</sup> Network File System  
<sup>۹</sup> Dynamic Provisioning  
<sup>۱۰</sup> Storage Class



شکل ۴-۴: نحوه کار PV و PVC در خوشه کوبرنتیز

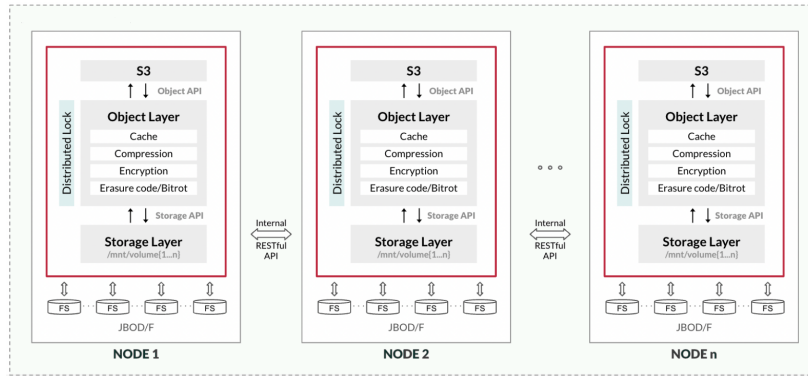
### ذخیره سازی داده

برای پلتفرم‌های MLOps که نیاز به ذخیره‌سازی حجم زیادی از داده‌ها دارند، استفاده از فضای ذخیره‌سازی ضروری است. ذخیره‌سازی شیء<sup>۱۱</sup> یک مدل ذخیره‌سازی داده است که برای مدیریت و دسترسی به داده‌های بدون ساختار مانند فایل‌های متنی، 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 میلیارد‌ها 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 ارائه‌ی 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 API های S3 را داراست، 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 خوشه، ظرفیت و عملکرد سیستم را به سادگی افزایش داد. در خوشه MinIO، داده‌ها به صورت خودکار توزیع و تکرار<sup>۱۲</sup> می‌شوند تا از دسترسی‌پذیری بالا و تحمل خطا اطمینان حاصل شود. یکی از مهم‌ترین تکنیک‌هایی که MinIO برای ذخیره‌سازی داده‌ها به کار می‌گیرد، کدگذاری

<sup>۱۱</sup> Object Storage

<sup>۱۲</sup> Replication



شکل ۴-۵: معماری MinIO

پاکسازی<sup>۱۳</sup> 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 باقی مانده بازیابی کند، بدون اینکه داده‌ای از دست برود. این روش نه تنها فضای ذخیره‌سازی را بهینه می‌کند بلکه تحمل خطا سیستم را نیز افزایش می‌دهد. این ویژگی‌ها MinIO را به انتخابی مناسب برای ذخیره‌سازی داده‌های حجیم 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 کرد و پیش پردازش‌های موردنظر را روی داده‌ها انجام داده و ویژگی‌های موردنظر را استخراج می‌کنند. در نهایت این ویژگی‌ها را می‌توانند در همان MinIO یا در پایگاه داده‌هایی که به همین منظور طراحی شده ذخیره نمایند.

### پایگاه داده

به منظور ذخیره‌سازی داده‌های ساختار یافته<sup>۱۴</sup> مانند داده‌های جدولی و یا ویژگی‌های بدست 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.

<sup>۱۳</sup> Erasure Coding

<sup>۱۴</sup> Structured

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 بدست 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<sup>۱۵</sup> 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 صف‌ها 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 نیاز مختلف سیستم را فراهم سازد.

## ۴-۳-۲ شبکه

بسیاری از برنامه‌های مدرن با استفاده از معماری میکروسرویس توزیع‌شده 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،

<sup>۱۵</sup>In-memory

ترافیک مدیریت شده، نظارت، و امنیت را بهبود می بخشد. این سیستم از پروکسی های جانبی<sup>۱۶</sup> برای کنترل ارتباطات بین میکروسرویس ها استفاده می کند، که این پروکسی ها معمولاً از Envoy، یک پروکسی سریع و سبک، بهره می برند [۴۸]. از ویژگی های مهم Istio می توان به سه مورد زیر اشاره کرد:

- مدیریت ترافیک: به کاربران اجازه می دهد تا ترافیک بین سرویس ها را به صورت دقیق کنترل و مدیریت کنند. با استفاده از قابلیت های مسیریابی پیشرفته، کاربران می توانند قوانین پیچیده ای برای مسیریابی ترافیک تعریف کنند. این قوانین شامل تقسیم بار<sup>۱۷</sup>، مسیریابی مبتنی بر نسخه (برای پیاده سازی به روزرسانی های متوالی)، و مدیریت ترافیک های خطا<sup>۱۸</sup> می شوند. این ویژگی ها به توسعه دهندگان کمک می کنند تا با اطمینان بیشتری به روزرسانی ها و تغییرات را در سیستم های خود اعمال کنند [۴۸].

- امنیت: امکانات امنیتی جامعی برای ارتباطات سرویس به سرویس فراهم می کند. این امکانات شامل احراز هویت<sup>۱۹</sup> و مجوزدهی<sup>۲۰</sup> مبتنی بر سیاست های امنیتی است. Istio با استفاده از MTLS ارتباطات بین سرویس ها را رمزنگاری می کند و اطمینان حاصل می کند که فقط سرویس های معتبر می توانند با یکدیگر ارتباط برقرار کنند. این قابلیت ها به افزایش امنیت سیستم های میکروسرویس کمک شایانی می کنند.

- مشاهده پذیری و نظارت: قابلیت های گسترده ای شامل جمع آوری و نمایش لاگ ها، متریک ها و تریس ها<sup>۲۱</sup> برای مشاهده پذیری و نظارت ارتباطات بین سرویس ها فراهم می کند. با استفاده از ابزارهای یکپارچه سازی شده مانند Prometheus و Grafana کاربران می توانند به صورت جامع عملکرد و سلامت سیستم های خود را نظارت کنند.

همان طور که معماری این سیستم را در شکل ۴-۶ می بینید، Istio به دو بخش اصلی سطح داده<sup>۲۲</sup> و سطح کنترل<sup>۲۳</sup> تقسیم می شود.

### سطح داده

سطح داده مدیریت ارتباط بین سرویس ها را بر عهده دارد. در یک شبکه سنتی بدون سرویس مش، شبکه نمی تواند ترافیک را بفهمد و بنابراین نمی تواند تصمیمات مبتنی بر نوع ترافیک یا منبع و مقصد آن بگیرد. با استفاده از این سیستم، هر ترافیک

<sup>۱۶</sup>Sidecar proxies

<sup>۱۷</sup>Load balancing

<sup>۱۸</sup>Fault injection

<sup>۱۹</sup>authentication

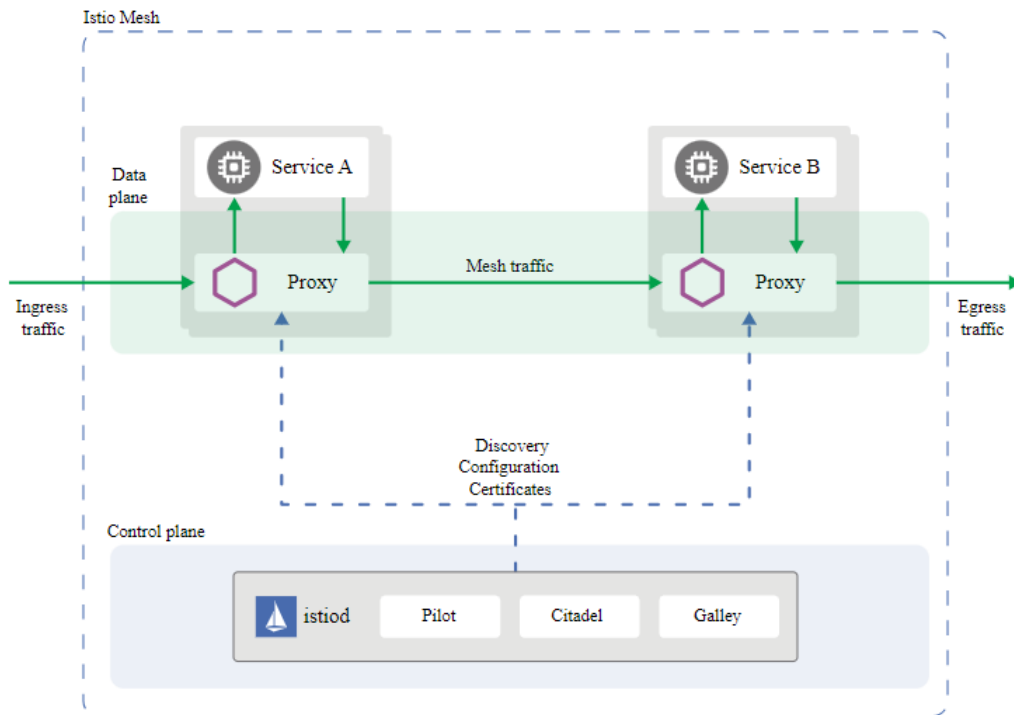
<sup>۲۰</sup>authorization

<sup>۲۱</sup>Trace

<sup>۲۲</sup>Data plane

<sup>۲۳</sup>Control plane





شکل ۴-۶: معماری Istio

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 پراکسی، به عنوان یک پروکسی با عملکرد بالا، قابلیت‌های گسترده‌ای 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 به تغییرات در توپولوژی پاسخ می‌دهد. تقسیم بار ترافیک را بهینه‌سازی می‌کند [۴۸]. این ابزار از پروتکل‌های HTTP/2 و gRPC برای بهبود کارایی پشتیبانی می‌کند و با استفاده از قطع‌کننده‌های مدار<sup>۲۴</sup> 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 تشکیل دهنده 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،

<sup>۲۴</sup>Circuit Breakers

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 بلادرنگ این امکان را می‌دهد تا سرویس‌ها 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<sup>۲۵</sup> 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<sup>۲۶</sup> 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 ایزوله‌سازی، 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<sup>۲۷</sup> و Istio، به همراه ابزار Dex برای مدیریت هویت<sup>۲۸</sup> 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.

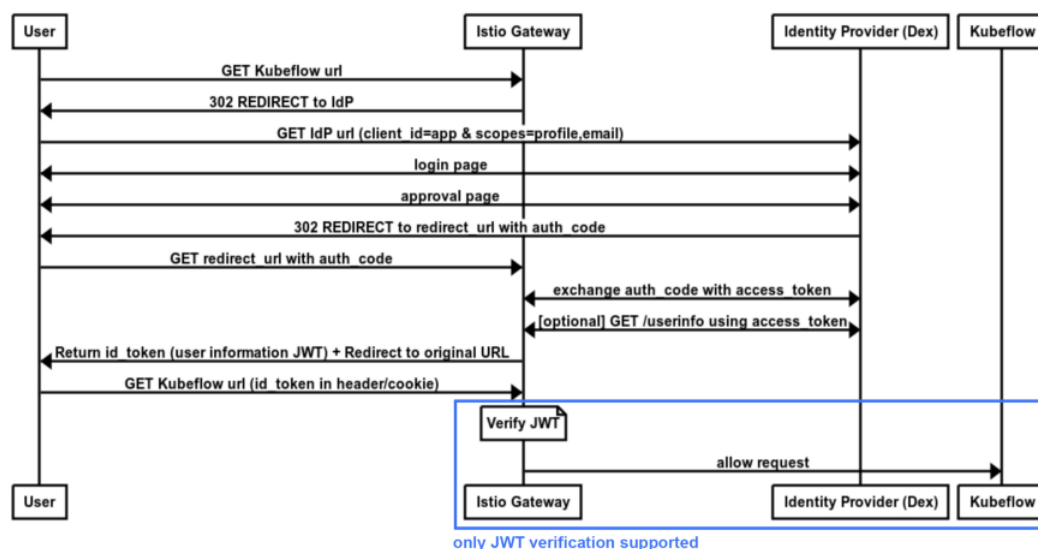
شکل ۴-۷ روند احراز هویت یک کاربر با استفاده از پروتکل OIDC را نشان می‌دهد. کاربران ابتدا به رابط کاربری پلتفرم ابری وارد می‌شوند. در این مرحله، Dex به عنوان ارائه‌دهنده هویت، صفحه ورود را به کاربر نشان می‌دهد. سپس Dex اطلاعات کاربر را از سیستم‌های احراز هویت خارجی می‌گیرد و پس از تأیید هویت، یک توکن 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

<sup>۲۵</sup>Certificate Authority

<sup>۲۶</sup>Multi-Tenancy

<sup>۲۷</sup>OpenID Connect

<sup>۲۸</sup>Identity Provider



شکل ۴-۷: روند احراز هویت

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.

## ایزوله‌سازی

ایزوله‌سازی در این پلتفرم با استفاده از namespace‌های کوبرنتیز انجام 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<sup>۲۹</sup> 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

<sup>۲۹</sup>Role-based access control

دسترسی‌های دقیقی برای کاربران تعیین کنند، مثلاً کاربری تنها بتواند خط لوله‌ها را مشاهده کند اما نتواند آنها را اجرا کند.

### ۴-۳-۴ نظارت

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

#### متریک‌های سیستم

متریک‌های سیستم اطلاعاتی درباره مصرف منابع مانند CPU، حافظه، استفاده از دیسک و ترافیک شبکه را شامل می‌شوند. این متریک‌ها به شناسایی مشکلات زیرساختی کمک می‌کنند که ممکن است بر عملکرد کلی سیستم تأثیر بگذارند. مثلاً افزایش مصرف CPU یا حافظه می‌تواند نشان‌دهنده بار غیرعادی یا مشکلات در کد مدل باشد.

#### متریک‌های مدل

متریک‌های مدل برای ارزیابی عملکرد مدل‌های یادگیری ماشین استفاده می‌شوند و شامل پارامترهایی مانند مدت زمان پیش‌بینی، دقت، F1-score و نرخ خطا هستند. مدت زمان پیش‌بینی نشان‌دهنده مدت زمان لازم برای انجام یک پیش‌بینی است و افزایش غیرعادی آن می‌تواند نشان‌دهنده مشکلات کارایی باشد. دقت مدل به ارزیابی کیفیت پیش‌بینی‌ها کمک می‌کند و کاهش در این متریک‌ها ممکن است نشان‌دهنده نیاز به بازآموزی مدل<sup>۳۰</sup> باشد. نرخ خطا نیز به شناسایی مشکلات احتمالی در داده‌ها یا الگوریتم‌ها کمک می‌کند.

#### متریک‌های داده

متریک‌های داده شامل کیفیت داده‌ها، تغییرات در توزیع داده‌ها<sup>۳۱</sup> و نرخ داده‌های مفقود می‌باشند. این متریک‌ها برای تضمین کیفیت داده‌های ورودی و جلوگیری از بروز مشکلات ناشی از داده‌های نادرست یا ناکافی ضروری هستند. مثلاً تغییرات ناگهانی در توزیع داده‌ها می‌تواند نشان‌دهنده تغییرات در رفتار کاربران یا نقص در فرآیند جمع‌آوری داده‌ها باشد. نرخ داده‌های مفقود نیز به شناسایی مشکلات در جریان داده‌ها کمک می‌کند و از تأثیرگذاری منفی بر عملکرد مدل جلوگیری می‌کند.

یکی از ابزارهای کلیدی برای جمع‌آوری این متریک‌ها Prometheus است که برای جمع‌آوری، ذخیره‌سازی و مدیریت متریک‌ها به کار می‌رود. Prometheus از یک مدل داده مبتنی بر سری‌های زمانی و یک زبان پرس‌وجوی قدرتمند به

<sup>۳۰</sup> Model Retraining  
<sup>۳۱</sup> Data Drift

نام PromQL استفاده می‌کند. Prometheus متریک‌ها را از طریق scraping از endpointهای مشخص شده جمع‌آوری می‌کند. 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 گیرندگان مناسب ارسال می‌کند. این گیرندگان می‌توانند شامل ایمیل‌ها، سرویس‌های پیام‌رسانی (مانند Slack) و یا حتی runbooksها برای اجرای خودکار اقدامات باشند. در نهایت، 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].

## ۴-۳-۵ استقرار مدل

## فصل ۵

# پیاده سازی و نتایج

## ۵-۱ پیاده سازی پلتفرم

### ۵-۱-۱ 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 سخت افزاری، پیاده‌سازی فرآیندهای مستمر بهبود و به‌روزرسانی، مدیریت دسترسی‌ها و امنیت و مستندسازی فرآیندها و تغییرات است. سیستم مدیریت شامل تمامی ابزارهای مدیریتی مانند مدیریت مخازن مولفه‌ها، کد و خط لوله CI/CD، مانیتورینگ کل سیستم و جمع‌آوری لاگ است. علاوه بر این، استراتژی استقرار پروژه، اعمال مهاجرت‌ها<sup>۱</sup>، پیکربندی پروژه‌ها، مدیریت سرورهای DNS و NTP<sup>۲</sup> و استفاده از ابزارهایی مانند Foreman برای نصب سیستم‌عامل 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 خطا<sup>۳</sup> و دسترسی پذیری بالا<sup>۴</sup> قرار می‌دهیم. از آنجایی که فرآیند و پروسه سنگینی روی این ماشین‌ها انجام نمی‌شود مشخصات کمتری می‌تواند به نسبت ماشین‌های پروژه داشته باشد. مشخصات سخت افزاری هرکدام از این ماشین‌ها در جدول ۵-۱ قرار دارد. این ماشین‌ها به صورت ماشین مجازی با استفاده از OpenStack ساخته می‌شود. ماشین‌های مدیریت باید برای نصب و راه‌اندازی

---

<sup>۱</sup>Migrations

<sup>۲</sup>Network Time Protocol

<sup>۳</sup>Fault Tolerance

<sup>۴</sup>High Availability

جدول ۵-۱: مشخصات سخت افزاری ماشین های مدیریت

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

ابزارهای مدیریت پلتفرم پیکربندی شوند و این کار با استفاده از ابزار Ansible انجام می گیرد. به این منظور Role های مشخصی برای هر بخش نوشته شده است تا بتوان بدون هیچ کار دستی و به صورت کاملاً خودکار سیستم ها را پیکربندی کرد. این Role های انسیبلی با استفاده از قاعده نسخه گذاری Semantic نسخه گذاری شده و در مخزن raw موجود در Nexus که یک ابزار مدیریت مخازن مولفه می باشد نگه داری خواهند شد. در نهایت برای پیکربندی سیستم، Role موردنظر با نسخه مشخص از Nexus گرفته شده و با استفاده از Ansible ماشین ها پیکربندی می شوند. از آنجایی که این پیکربندی در محیط های مختلف مانند توسعه و عملیات می تواند متفاوت باشد، ما با استفاده از قابلیت Overriding در این ابزار مقادیر پیش فرض را برای هر محیط تغییر خواهیم داد. به همین منظور اسکریپتی طراحی شده که در لینک گیت هاب<sup>۵</sup> قابل مشاهده است. پروسه پیکربندی و نصب ابزار در ماشین های مدیریت به صورت زیر انجام می گردد:

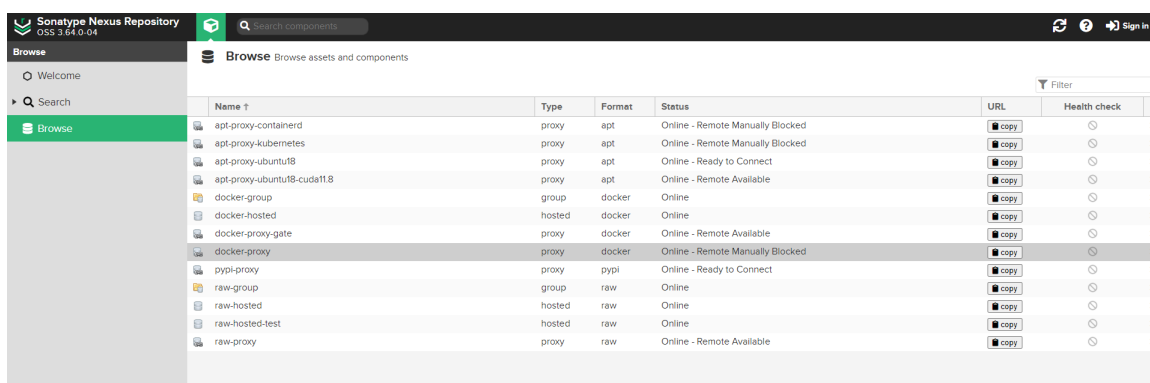
۱. پیکربندی ماشین ها: این قسمت شامل نصب و پیکربندی ابزارهایی نظیر BIND برای سرور DNS، APT برای مدیریت ابزار در سیستم عامل Ubuntu، pip برای مدیریت کتابخانه ها پایتون، chrony برای سرور NTP، LDAP برای مدیریت کاربران و ... می باشد.

۲. نصب و پیکربندی Docker: از آنجایی که مدیریت ابزارها به صورت کانتینر مناسب تر است، برای انجام مراحل بعدی نیاز به نصب Docker می باشد. پس از نصب به منظور ذخیره سازی تمام مولفه ها مورد استفاده به مخزن ساخته شده در Nexus مدیریت متصل خواهد شد.

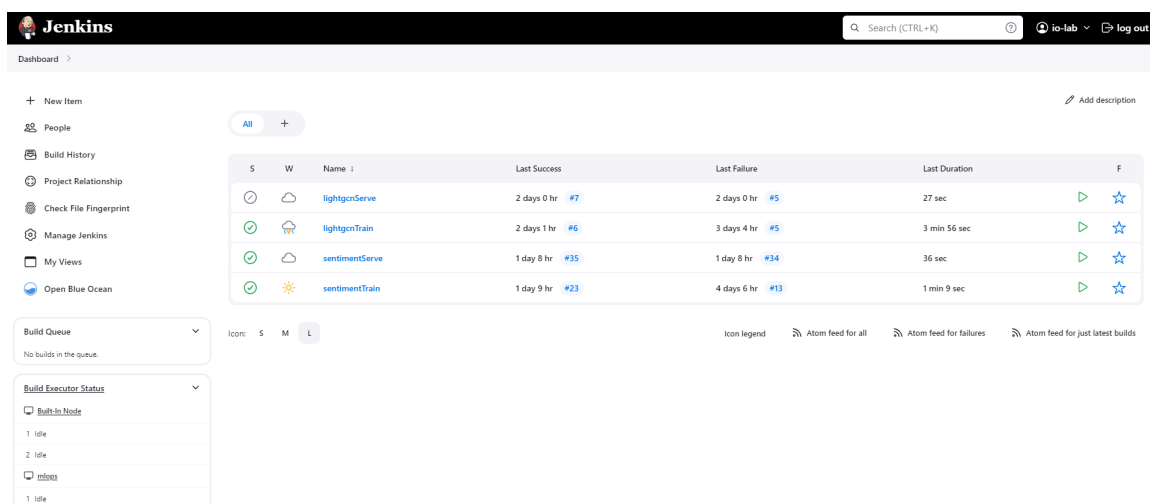
۳. Nexus: از این ابزار به منظور مدیریت مخازن مولفه ها استفاده شده است. مخازن مورد استفاده ما APT، pip، Docker و raw می باشد (شکل ۵-۱).

۴. Jenkins: از این ابزار به منظور اجرا و مدیریت خط لوله های CI/CD پروژه ها و هم چنین پیکربندی آن ها توسط مدیران سیستم استفاده می شود (شکل ۵-۲).

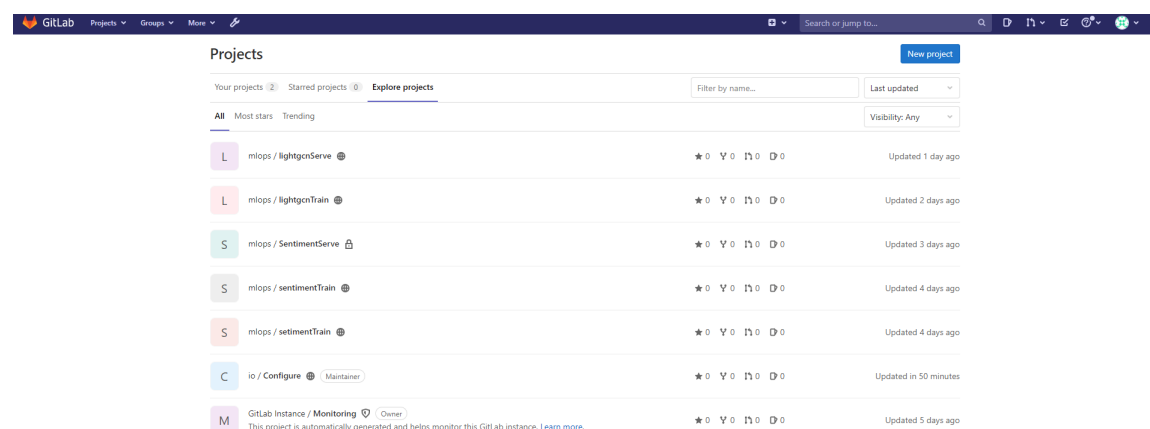
۵. GitLab: به منظور مدیریت کد در پروژه ها و هم چنین مدیریت Role های انسیبلی برای پیکربندی پروژه ها استفاده می شود (شکل ۵-۳).



شکل ۵-۱: مخازن مولفه در Nexus



شکل ۵-۲: خط لوله ها CI/CD در Jenkins



شکل ۵-۳: مخازن کد در Gitlab



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

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

## ۵-۱-۲ خوشه کوبرنتیز

در طراحی یک پلتفرم MLOps جامع و کارآمد که تمامی ابزارهای مورد نیاز را در بر می گیرد، هدف اصلی ایجاد یک بستر یکپارچه، مقیاس پذیر و انعطاف پذیر برای مدیریت چرخه حیات مدل های یادگیری ماشین است. این پلتفرم شامل مجموعه ای از ابزارها و تکنولوژی های متن باز است که همگی روی خوشه کوبرنتیز مستقر می شوند. استفاده از کوبرنتیز در پلتفرم های MLOps به دلیل قابلیت های منحصر به فرد آن در مدیریت خودکار، مقیاس پذیری و کارایی منابع است.

برای نصب خوشه کوبرنتیز و انجام تست های اولیه پروژه MLOps، ۴ ماشین مجازی با مشخصات ذکر شده در جدول ۵-۲ استفاده شده است. در این پیاده سازی سه ماشین اول به عنوان گره اصلی و ماشین چهارم به عنوان گره کاری انتخاب شده است. همانند سیستم های مدیریت، ابتدا ماشین های مجازی که توسط OpenStack ساخته خواهند شد پیکربندی می شوند. پس از آن با استفاده از ابزار Kubeadm خوشه کوبرنتیز پیاده سازی خواهد شد.

## پیاده سازی خوشه کوبرنتیز

برای نصب و پیاده سازی کوبرنتیز با استفاده از kubeadm، ابتدا پیش نیازهایی مانند Containerd و بسته های کوبرنتیز شامل kubelet، kubeadm و kubectl آماده می شوند. پس از پیکربندی سیستم و نصب Containerd، بسته های مورد نیاز نصب شده و Swap غیرفعال می گردد، چرا که کوبرنتیز برای عملکرد بهینه نیاز به غیرفعال بودن Swap دارد. راه اندازی خوشه با اجرای دستور kubeadm init بر روی گره اصلی آغاز می گردد. این دستور تنظیمات اولیه را انجام داده و فایل پیکربندی برای kubectl ایجاد می کند که باید برای دسترسی به خوشه تنظیم گردد. پس از آن، نصب رابط شبکه کانتنر<sup>۶</sup> انجام می شود و در این مورد، ابزار Calico مورد استفاده قرار می گیرد. نصب Calico امکان برقراری ارتباط بین پادها در داخل خوشه را فراهم می سازد.

در مرحله بعد، گره های کارگر به خوشه اضافه می شوند. دستورات لازم برای پیوستن نودهای کارگر به خوشه که از دستور قبل به دست آمده اند، بر روی هر گره کارگر اجرا می گردد تا این گره ها به گره اصلی متصل شوند. در نهایت، وضعیت گره ها و صحت پیوستن آن ها به خوشه با استفاده از kubectl get nodes بررسی می گردد و اطمینان حاصل می شود که همه گره ها به درستی اضافه شده و آماده به کار هستند. این روش نصب، راهی سریع و مؤثر برای راه اندازی خوشه کوبرنتیز فراهم می آورد.

<sup>۶</sup> Container Network Interface (CNI)

که امکان اضافه کردن گره های جدید و مدیریت بهتر و مقیاس پذیری را به سهولت فراهم می سازد. تمامی این فرآیند با استفاده از Role های انسیبلی روی ماشین های پروژه انجام می گردد.

### پیاده سازی مولفه های پروژه

تمامی مولفه ها انتخاب شده در طراحی معماری فصل قبل با استفاده از Helm که وظیفه مدیریت مولفه ها را دارد انجام می گیرد. Helm یک ابزار قدرتمند برای مدیریت بسته های نرم افزاری کوبرنیتیز است که فرآیند استقرار، بروزرسانی و مدیریت برنامه ها را ساده تر می کند. نصب یک چارت با استفاده از Helm این امکان را می دهد که برنامه ها و سرویس ها با کمترین تلاش ممکن و به صورت اتوماتیک در خوشه کوبرنیتیز مستقر شوند. چارت مورد نظر با استفاده از دستور helm install به راحتی نصب می گردد. این دستور، چارت مورد نظر را از مخازن موجود در Nexus دانلود کرده و آن را در خوشه کوبرنیتیز مستقر می کند. این کار نیز با استفاده از یک Role انسیبلی که نوشته شده است انجام می گیرد.

یکی از مزایای استفاده از Helm، قابلیت تنظیم پارامترهای چارت ها در زمان نصب است. این قابلیت امکان سفارشی سازی چارت ها را براساس نیازهای خاص پروژه فراهم می کند. با استفاده از این ویژگی و قابلیت مشابه در انسیبل، همانند قبل برای هر محیط مقادیر خاص آن محیط را پیکربندی می کنیم. علاوه بر این، Helm با ارائه قابلیت بازگشت پذیری، امکان بازگشت به نسخه های قبلی چارت ها را نیز فراهم می سازد که این ویژگی برای مدیریت نسخه ها و رفع مشکلات بسیار مفید است.

تمامی مولفه های اصلی نصب شده روی خوشه کوبرنیتیز به منظور پیاده سازی پلتفرم MLOps در شکل ۴-۵ نشان داده شده است. نکته ای که راجع به این مولفه ها لازم به ذکر است، به منظور افزایش قابلیت اطمینان و تحمل خطا، استفاده بیش از یک replica برای هر مولفه ضروری است. این امر باعث می شود تا در صورت خرابی یکی از نمونه ها، مولفه مربوطه به کار خود ادامه دهد و سرویس دهی دچار اختلال نشود. در شکل ۴-۵ نیز میزان مصرف منابع سخت افزاری سیستم زمانی که پلتفرم زیر بار نیست نشان داده شده است. همان طور که می بینید میزان مصرف منابع سخت افزاری به صورت یکنواخت به خوبی توزیع شده است که منجر به استفاده بهینه از منابع سخت افزاری می گردد.

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 ۴-۵: میزان مصرف منابع سخت افزاری پلتفرم بدون بار

## ۵-۲ حل مسئله و نتایج

## فصل ۴

# نتیجه‌گیری و پیشنهادات

با پیشرفت فن‌آوری دیجیتال و گسترش هرچه بیشتر کاربردهای سرویسهای چندرسانه‌ای دیجیتال، نیازهای امنیتی جدیدی در سطح جهان مطرح گردیده است و لذا با نفوذ دنیای دیجیتال به زندگی مردم، طراحی سیستمهای امنیتی مرتبط به آن اهمیت فراوانی در سالهای اخیر پیدا کرده‌اند. به دنبال این نیاز، نهان‌نگاری به عنوان روشی مؤثر جهت تأمین برخی از این نیازها مورد توجه قرار گرفته و پیشرفت سریعی داشته است.

در این پایان‌نامه جهت‌آشنایی و نیل به یک دیدگاه کلی از سیستمهای نهان‌نگاری ابتدا به بیان کاربردهای نهان‌نگاری

پرداختیم. ...

# مراجع

- [1] L Navarro E Hernandez-sanchez F Rodriguez-Haro, F Freitag, “A summary of virtualization techniques,” .
- [2] 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.
- [3] Inc. Amazon Web Services, “What is devops?,” URL: <https://aws.amazon.com/devops/what-is-devops/> [Accessed: 2024-05-07].
- [4] 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.
- [5] 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].
- [6] Jez Humble and David Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, Addison-Wesley Professional, 2010.
- [7] paul hammant, “Trunk based development,” URL: <https://trunkbaseddevelopment.com/> [Accessed: 2023-11-01].
- [8] M. Huttermann, *DevOps for Developers*, chapter Infrastructure as Code, 2012.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] Anja Kammer Florian Beetz and Dr. Simon Harrer, *GitOps Cloud-native Continuous Deployment*, 2021.
- [13] N. Forsgren and J. Humble, “The role of continuous delivery in it and organizational performance,” March 2016.
- [14] “What is virtualization?,” URL: <https://aws.amazon.com/what-is/virtualization/> [Accessed: 2024-05-08].
- [15] 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.
- [16] “Containerization,” URL: <https://www.ibm.com/topics/containerization> [Accessed: 2023-05-21].

- [17] Docker Inc., “Docker,” URL: <https://docs.docker.com/> [Accessed: 2023-05-18].
- [18] Y. Yu Y. Zhou and B. Ding, “Towards mlops: A case study of ml pipeline platform,” October 2020.
- [19] Damian A. Tamburri, “Sustainable mlops: Trends and challenges,” in *2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNAS)*, 2020, pp. 17–23.
- [20] 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.
- [21] Ioannis Karamitsos, Saeed Albarhami, and Charalampos Apostolopoulos, “Applying devops practices of continuous automation for machine learning,” *Information*, vol. 11, no. 7, 2020.
- [22] 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.
- [23] Behrouz Derakhshan, Alireza Rezaei Mahdiraji, Tilmann Rabl, and Volker Markl, “Continuous deployment of machine learning pipelines,” in *EDBT*, March 2019, pp. 397–408.
- [24] Alexandra Posoldova, “Machine learning pipelines: From research to production,” *IEEE Potentials*, vol. 39, no. 6, pp. 38–42, 2020.
- [25] “Apache airflow,” URL: <https://airflow.apache.org/> [Accessed: 2023-11-02].
- [26] “Kubeflow,” URL: <https://www.kubeflow.org/> [Accessed: 2023-11-02].
- [27] 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].
- [28] “Feast,” URL: <https://feast.dev/> [Accessed: 2023-11-05].
- [29] Álvaro López García, Jesús Marco De Lucas, Marica Antonacci, Wolfgang Zu Castell, Mario David, Marcus Hardt, Lara Lloret Iglesias, Germán Moltó, 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, Zdeněk Šustr, and Pawel Wolniewicz, “A cloud-based framework for machine learning workloads and applications,” *IEEE Access*, vol. 8, pp. 18681–18692, 2020.
- [30] “Mlflow,” URL: <https://mlflow.org/> [Accessed: 2023-11-21].
- [31] “Apache spark,” URL: <https://spark.apache.org/> [Accessed: 2023-11-29].
- [32] “Knative,” URL: <https://knative.dev/docs/> [Accessed: 2023-11-29].
- [33] 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].
- [34] “Gitlab,” URL: <https://about.gitlab.com/> [Accessed: 2024-05-16].
- [35] “Gerritcodereview,” URL: <https://www.gerritcodereview.com/> [Accessed: 2024-05-16].
- [36] “Jenkins,” URL: <https://www.jenkins.io/> [Accessed: 2023-11-01].
- [37] Armin Esmailzadeh, Maryam Heidari, Reyhaneh Abdolazimi, Parisa Hajibabaei, and Masoud Malekzadeh, “Efficient large scale nlp feature engineering with apache spark,” in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, 2022, pp. 0274–0280.
- [38] Jun Xu, “Mlops in the financial industry: Philosophy, practices, and tools,” in *Future and Fintech, the, Abcdi and Beyond*, p. 451. World Scientific, 2022.
- [39] “Kafka,” URL: <https://kafka.apache.org/> [Accessed: 2023-11-31].

- [40] “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] “Ansible,” URL: <https://www.ansible.com/> [Accessed: 2023-12-01].
- [43] “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,” 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].

## **ABSTRACT**

In the digital world today, invisible and robust image watermarking which embeds invisible signals in to the digital images has been proposed as a major solution to the problem of copyright protection of digital images. Several approaches such as exploiting Human Visual System (HVS) and invariant domain watermarking have been proposed to achieve this goal. In this thesis we use the information-theoretic concepts as tools to develop methods for embedding watermark in an optimized way. Also multi-resolution transforms such as wavelet transform and MR-SVD (Multi-Resolution form of the Singular Value Decomposition) are used in the proposed structure, because theses transforms resemble the HVS characteristics for an optimized watermarking structure. Entropy concept and entropy masking effects were proposed to use to develop a model in DWT domain to increase the strength and robustness of the watermark, while perceived quality of the electronic image is not altered. Then, the structure similar to the entropy-based proposed structure in DWT domain, is used for watermarking in the MR-SVD transform domain, which is found a new approach to robust image watermarking. Simulation results show that the proposed methods outperform conventional methods in terms of both invisibility and robustness.

## **KEYWORDS**

1. Image Watermarking.
2. Multi-Resolution Transform.
3. Human Visual System (HVS).
4. Wavelet Transform.
5. Singular Value Decomposition (SVD).
6. Entropy.
7. Entropy Masking.





SHARIF UNIVERSITY OF TECHNOLOGY  
ELECTRICAL ENGINEERING DEPARTMENT

M.Sc. THESIS

*Title:*

# An Information-Theoretic Model for Image Watermarking

*by:*

AAAAA BBBB

*Supervisor:*

Dr. ...

August 2005