کنترل‌های قبل از دیپلوی برای مدیریت ریسک تنظیمات محیطی و متغیرهای `env`

مقدمه

در هر سیستم CI/CD، کنترل‌های قبل از دیپلوی (Pre-Deployment Controls) نقش حیاتی در مدیریت ریسک‌های مرتبط با تنظیمات محیطی و متغیرهای محیطی (`env`) ایفا می‌کنند. این کنترل‌ها تضمین می‌کنند که نرم‌افزار به درستی، ایمن و با پیکربندی صحیح در هر محیط مستقر شود. در سیستم CI/CD پیاده‌سازی شده برای مونوریپو `aladdin-sandbox` ، این کنترل‌ها به صورت چندلایه و با استفاده از قابلیت‌های `mamos_runner.py` و GitHub Actions Workflows اعمال می‌شوند.

1. تفکیک محیط‌ها و پیکربندی‌ها

اساس مدیریت ریسک محیطی، تفکیک واضح بین محیط‌های مختلف است. در سیستم ما، محیط‌های زیر به صورت منطقی و پیکربندی شده تعریف شده‌اند:

- **محیط توسعه (Development Environment)**: این محیط معمولاً محلی است و توسط توسعه‌دهندگان برای کدنویسی و تست‌های اولیه استفاده می‌شود. پیکربندی‌ها در این محیط ممکن است به صورت محلی در فایل‌های `env.` یا تنظیمات IDE نگهداری شوند.
- **محیط تست (Test Environment)**: محیطی که در آن تست‌های خودکار (واحد، یکپارچه‌سازی) و گاهی تست‌های دستی اجرا می‌شوند. این محیط توسط خط لوله CI/CD مدیریت می‌شود.
- **محیط استیجینگ (Staging Environment)**: یک محیط شبیه‌سازی شده از پروداکشن که برای تست‌های جامع‌تر، تست‌های پذیرش کاربر (UAT) و نمایش به ذینفعان استفاده می‌شود. این محیط تا حد امکان باید مشابه پروداکشن باشد.
- **محیط پروداکشن (Production Environment)**: محیط زنده که کاربران نهایی از محصول استفاده می‌کنند. این محیط دارای بالاترین سطح کنترل و امنیت است.

2. مدیریت تنظیمات محیطی از طریق `config/projects.yaml`

فایل `config/projects.yaml` نقش کلیدی در تعریف تنظیمات مربوط به استقرار هر پروژه در محیط‌های مختلف دارد:

- **شناسه‌های سرویس Render.com**: برای هر پروژه و هر محیط استقرار (Test, Staging, Production)، یک `render_service_id` منحصر به فرد تعریف شده است. این شناسه‌ها به `mamos_runner.py` اجازه می‌دهند تا بداند کدام سرویس در Render.com مربوط به کدام پروژه و محیط است. این تفکیک تضمین می‌کند که استقرارها به صورت هدفمند و بدون تداخل انجام شوند.
- **دستورات Build و Test**: دستورات `build` و `test` نیز در این فایل برای هر پروژه تعریف شده‌اند. این دستورات ممکن است شامل نصب وابستگی‌ها یا اجرای اسکریپت‌هایی باشند که بسته به محیط، رفتار

متفاوتی دارند (مثلاً نصب وابستگی‌های توسعه در محیط توسعه و وابستگی‌های پروداکشن در محیط پروداکشن).

3. مدیریت متغیرهای محیطی (`env`) و Secrets

متغیرهای محیطی، به ویژه آن‌هایی که حاوی اطلاعات حساس (مانند کلیدهای API، رمزهای عبور پایگاه داده) هستند، باید با دقت بالا مدیریت شوند. سیستم ما از قابلیت‌های GitHub Actions برای این منظور استفاده می‌کند:

- **GitHub Secrets**: تمامی متغیرهای محیطی حساس (مانند `RENDER_API_KEY`) به عنوان GitHub Secrets ذخیره می‌شوند. این روش تضمین می‌کند که این اطلاعات هرگز در کد یا فایل‌های پیکربندی در دسترس هستند workflow عمومی قرار نمی‌گیرند و فقط در زمان اجرای.
- **تخصیص متغیرهای محیطی در Workflow**: در فایل‌های GitHub Actions Workflow، متغیرهای محیطی مورد نیاز برای هر Job یا Step به صراحت تعریف می‌شوند. این متغیرها می‌توانند از GitHub Secrets یا از مقادیر ثابت تامین شوند.

4. کنترل‌های قبل از دیپلوی در GitHub Actions Workflows

GitHub Actions Workflows شامل چندین کنترل داخلی برای مدیریت ریسک‌های قبل از دیپلوی است:

- **تریگرهای مبتنی بر مسیر (Path-based Triggers)**: همانطور که قبلاً ذکر شد، workflowها تنها زمانی فعال می‌شوند که تغییراتی در مسیرهای مربوط به یک پروژه خاص رخ دهد. این امر از اجرای غیرضروری و بالقوه خطرناک استقرار برای پروژه‌هایی که تغییری نکرده‌اند، جلوگیری می‌کند.
- **محیط‌های GitHub Actions (Environments)**: این قابلیت به ما اجازه می‌دهد تا محیط‌های استقرار (مانند `Staging` و `Production`) را تعریف کرده و قوانین محافظت (Protection Rules) را برای آن‌ها اعمال کنیم. این قوانین شامل:
- **تایید دستی (Manual Approval)**: مهمترین کنترل قبل از دیپلوی، نیاز به تایید دستی است. برای محیط‌های `Staging` و `Production` ، یک یا چند کاربر/تیم مشخص باید به صورت دستی استقرار را تایید کنند. این امر یک لایه انسانی برای بازبینی نهایی تنظیمات، بررسی نتایج تست‌ها و اطمینان از آمادگی محیط فراهم می‌کند.
- **متغیرهای محیطی خاص محیط**: می‌توان متغیرهای محیطی خاصی را برای هر محیط در GitHub Actions تعریف کرد که فقط در آن محیط در دسترس باشند. این امر به جداسازی کامل پیکربندی‌ها کمک می‌کند.
- **بررسی وضعیت CI (CI Status Checks)**: قبل از اینکه یک Job استقرار (Deploy Job) آغاز شود، می‌توان Jobهای قبلی (مانند `build` و `test`) را به عنوان پیش‌نیاز تعریف کرد. این بدان معناست که استقرار تنها در صورتی آغاز می‌شود که تمامی مراحل ساخت و تست با موفقیت انجام شده باشند و هیچ خطایی وجود نداشته باشد.

5. نقش `mamos_runner.py` در کنترل‌ها

کنترل‌های زیر را اعمال می‌کند، CI/CD به عنوان عامل اجرایی در خط لوله `mamos_runner.py`:

- **اعتبارسنجی پیکربندی**: این اسکریپت فایل `config/projects.yaml` را می‌خواند و اعتبارسنجی‌های اولیه را بر روی پیکربندی پروژه و محیط انجام می‌دهد تا از صحت اطلاعات اطمینان حاصل کند.
- **مدیریت خطا**: `mamos_runner.py` به گونه‌ای طراحی شده است که خطاهای مربوط به دستورات shell و فراخوانی‌های API را به درستی مدیریت کند. در صورت بروز خطا در هر مرحله (مانند خطای `401` `Client Error: Unauthorized` در شبیه‌سازی استقرار)، فرآیند متوقف شده و گزارش خطا ارائه می‌شود، که از استقرار نسخه‌های ناقص یا مشکل‌دار جلوگیری می‌کند.
- **فراخوانی API استقرار**: این اسکریپت مسئول فراخوانی API پلتفرم استقرار (مانند Render.com) با استفاده از `render_service_id` و `RENDER_API_KEY` مناسب برای محیط هدف است. این فراخوانی‌ها تحت کنترل‌های امنیتی و دسترسی API پلتفرم ابری قرار دارند.

نتیجه‌گیری

سیستم CI/CD پیاده‌سازی شده با ترکیب تفکیک محیطی، مدیریت متمرکز پیکربندی‌ها در `config/projects.yaml` ، استفاده ایمن از متغیرهای محیطی و Secrets در GitHub Actions، و اعمال کنترل‌های فرآیندی مانند تایید دستی و بررسی وضعیت CI، یک رویکرد قوی برای مدیریت ریسک‌های مرتبط با تنظیمات محیطی و متغیرهای `env` قبل از دیپلوی ارائه می‌دهد. این کنترل‌ها به تیم توسعه اطمینان می‌دهند که هر استقرار به صورت ایمن، صحیح و با حداقل ریسک انجام می‌شود.