# haizhouyuan/storyapp 与 haizhouyuan/points 容器化及 CI 集成迁移方案

## 差异对比分析

StoryApp 项目已实现前后端**一体化容器部署**和**CI/CD 工作流**，而 Points 项目目前仅为前端应用，尚未配置容器化和持续集成。下表对比了两者在架构、容器配置、CI 集成等方面的差异：

| **对比维度** | **StoryApp（现状）** | **Points（现状）** |
| --- | --- | --- |
| **项目架构** | 前后端+数据库的全栈应用（React 前端、Node.js Express 后端、MongoDB 数据库）[[1]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/README.md#L14-L19) | 纯前端单页应用（React + Vite 实现的游戏化界面）[[2]](https://github.com/haizhouyuan/points/blob/69a6ac25bc55e4b40ca406de323d74172ff666f3/README.md#L2-L5) |
| **Dockerfile** | 提供多阶段 Dockerfile，将前端和后端统一构建到一个镜像中[[3]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L2-L10)（分阶段构建前端静态资源、后端代码，再组装运行环境） | 无 Dockerfile（尚未容器化；需要新增用于前端构建和静态服务的 Dockerfile） |
| **Docker Compose** | 使用 docker-compose 编排多个服务：定义应用服务(app)和 MongoDB 数据库服务[[4]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L22-L30)；公共网络storyapp-net[[5]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L6-L14)以及持久化卷（如 mongo\_data、app\_logs 等）[[6]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L11-L19) | 无 Compose 配置（暂无多容器需求；Points 无后端或数据库服务） |
| **服务配置** | Compose 中 app 服务通过 .env 加载环境变量[[7]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L44-L50)，依赖 MongoDB（depends\_on 等）[[7]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L44-L50)；开放端口 5000（可通过环境变量配置）[[8]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L52-L60)；MongoDB 服务设置默认管理员账号密码[[9]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L26-L34) | 无服务定义（仅需一个前端静态服务容器；Points 不涉及后台依赖） |
| **环境变量** | 使用**环境文件**管理配置：提供 .env.example 模板，包含 DeepSeek API 密钥、MongoDB 连接等配置项[[10]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/.env.example#L4-L12)；容器启动时自动加载 .env[[7]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L44-L50)（如 MONGODB\_URI 等） | 未使用环境变量配置（无 .env 文件；前端应用暂不涉及敏感配置，所有数据硬编码在前端） |
| **日志与持久化** | 挂载卷保存日志和上传内容：定义 app\_logs、app\_uploads 卷映射到容器内目录，持久化应用日志和上传文件[[8]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L52-L60)；支持配置开启详细日志、数据库日志等[[11]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/.env.example#L20-L28) | 无日志持久化配置（Points 为前端 UI，无服务器日志；暂不需要卷映射） |
| **CI 工作流** | 配置了多套 GitHub Actions 工作流：每次 PR 触发 CI 流水线进行代码质量检查（Lint、类型检查、测试、E2E 等）[[12]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L12-L20)；合并到主分支触发构建 Docker 镜像并推送到 GHCR[[13]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L21-L28)；提供手动触发的部署工作流实现一键部署生产环境[[14]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L29-L37) | 无任何 CI/CD 工作流（需要从零建立与 StoryApp 类似的 CI 流程，包括构建、测试、发布等） |
| **镜像发布** | 使用 GitHub Container Registry (GHCR) 托管 Docker 镜像。CI 工作流中自动将镜像推送到 ghcr.io[[15]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L6-L14)；采用多标签策略（如 sha-latest、提交短 SHA、版本号等）标识镜像[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318) | 未发布过容器镜像（需规划使用 GHCR 等统一镜像仓库，制定镜像命名和推送策略） |
| **安全机制** | 强调安全和维护性：容器内以非 root 用户 (storyapp) 运行应用，降低权限[[17]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L55-L63)；MongoDB 在生产/验证环境启用认证且不暴露端口[[18]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L24-L31)；敏感信息通过 Secrets 管理（如 Claude API Key、SSH 私钥等）[[19]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L39-L48)；健康检查端点 /healthz 监控服务存活[[20]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L61-L67) | 尚未配置专门的安全措施（容器迁移需考虑运行用户权限、Secrets 管理等；前端静态服务无内置健康检查，可通过 HTTP 状态检查代替） |

*表：StoryApp 与 Points 项目容器化和 CI 集成差异对比*

上述对比显示，StoryApp 已经构建了完善的容器化和自动化流程，而 Points 目前没有相应机制。尤其是 StoryApp 采用了**多阶段构建**（将前后端一起打包)、**多服务编排**（应用+数据库）、**多环境 Compose 文件**（开发、CI、生产）、**CI/CD 工作流**（持续集成、自动构建推镜像、按需部署）等一系列最佳实践。这些都是 Points 项目在迁移时需要借鉴和引入的。

此外，StoryApp 还在**安全**与**维护**上做了特殊考虑，例如通过 .env 文件和 GitHub Secrets 管理 API 密钥等敏感信息[[19]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L39-L48)、Dockerfile 中使用非特权用户运行[[17]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L55-L63)、只在开发环境开放数据库无密码访问[[21]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.dev.yml#L44-L52)而生产环境收紧访问控制等。相比之下，Points 由于仅为前端应用，目前不存在敏感后端配置，但在移植容器与CI时也应引入类似的安全措施（详见后文建议）。

## 迁移方案与步骤（分阶段）

针对上述差异，下面制定一个分阶段的迁移与适配计划，将 StoryApp 的容器化部署和 CI/CD 集成迁移到 Points 项目：

1. **容器化配置移植**：首先在 Points 项目中新增 Docker 配置。创建**Dockerfile**，采用多阶段构建：用 Node.js 镜像安装依赖并构建前端静态文件，然后用轻量级服务器镜像（如 Nginx）部署构建产物，实现前端静态资源的容器化。确保构建输出目录与 Points 项目构建产物匹配（Points 使用 Vite，默认构建输出目录为 build[[22]](https://github.com/haizhouyuan/points/blob/69a6ac25bc55e4b40ca406de323d74172ff666f3/vite.config.ts#L52-L60)）。同时，设置容器启动时的工作目录、端口等参数，并遵循 StoryApp 中的最佳实践，例如运行非 root 用户等。
2. **Docker Compose 编排适配**：在 Points 项目中新建**docker-compose.yml**，整合同步容器服务。由于 Points 无需后端数据库，仅需定义一个前端服务容器。在 Compose 中配置网络（例如 points-net）以隔离容器网络环境[[5]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L6-L14)，服务名称和容器名称（如 points-app），以及端口映射等。可以参考 StoryApp Compose 基座配置，将 Points 容器的**端口**通过环境变量配置，避免与 StoryApp 冲突（例如默认将 Points 容器内部 80 端口映射到主机的 5001 端口)。[[8]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L52-L60)由于 Points 不涉及数据库，可省略数据库服务和相关卷。若需要**开发模式**容器，可另行编写 docker-compose.dev.yml 覆盖文件，实现源码挂载和热更新（Points 开发环境通常直接运行 npm run dev 即可，但也可仿照 StoryApp 在容器中运行开发服务器，监听 3000 端口）。
3. **环境变量与配置迁移**：在 Points 项目根目录添加**.env.example**模板，用于日后管理配置。虽然当前 Points 前端应用暂无后端 API 或敏感配置，可以先保留一些基本变量（例如 NODE\_ENV=production、APP\_PORT=5001 等）以供容器和 Compose 使用。如果将来 Points 需要与后端服务交互，可按 StoryApp 模式加入诸如 API\_BASE\_URL 等环境变量，并通过 Vite 的前缀机制（如 VITE\_API\_URL）注入前端。在 GitHub 仓库中，准备好所需的 Secrets，例如如果部署需要 SSH 密钥、GHCR 登录令牌等（Points 项目目前没有外部 API 密钥，但部署阶段需配置类似 StoryApp 的 PROD\_HOST、PROD\_USER、PROD\_SSH\_KEY、GHCR\_PAT 等机密信息[[19]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L39-L48)）。
4. **CI 工作流集成**：参考 StoryApp 的 CI 工作流，在 Points 仓库下的**.github/workflows**目录中新建 CI YAML 配置。例如，创建ci.yml，在每次 Pull Request 时触发，对 Points 前端代码执行基础的质量检查和构建测试[[12]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L12-L20)。由于 Points 主要是前端项目，可在 CI 中执行步骤包括：安装依赖（npm ci）、构建（npm run build）以确保代码能成功打包，如有需要也可加入代码风格检查或简单测试（若项目有对应脚本）。这一阶段主要保证 Points 项目的代码在集成时不会破坏构建。(*注：StoryApp CI 还包含了更复杂的检查如 E2E 测试和 AI 代码审查*[*[23]*](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L16-L19)*。如果 Points 项目后续规模扩大，可考虑逐步加入，但初期可先实现基本的构建与检查。*)
5. **Docker 镜像构建与推送**：创建**Docker 构建推送工作流**（如docker-build-push.yml），实现在代码合并主分支或发布 tag 时自动构建并发布 Points 的 Docker 镜像[[13]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L21-L28)。流程应包括：使用 GitHub 提供的 GITHUB\_TOKEN 登录 GHCR[[24]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L291-L299)；构建 Points 项目的 Docker 镜像（利用上面编写的 Dockerfile，多阶段构建前端)；为镜像打上标签并推送到 GitHub Container Registry。[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318)推荐采用与 StoryApp 一致的**镜像命名规范**：例如 ghcr.io/haizhouyuan/points:sha-<Git短SHA> 表示特定提交镜像，ghcr.io/haizhouyuan/points:sha-latest 指向最新一次构建，以及将来可用语义化版本标签等。[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318) 推送完成后，可在工作流中加入**镜像运行验证**步骤：启动刚推送的 Points 镜像容器，在本地端口上试探性请求静态页面，确认返回 200 状态，确保镜像可正常运行（类似 StoryApp 中在推送后进行健康检查验证[[25]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L300-L305)，但由于 Points 无后端 API，可改为简单请求首页）。
6. **部署流程集成**：根据 StoryApp 生产部署方案，建立 Points 的**部署工作流**（如deploy-prod.yml）。该工作流可使用手动触发（workflow\_dispatch），在需要部署时运行。部署步骤与 StoryApp 类似：通过 SSH 连接目标服务器[[14]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L29-L37)（使用预先配置在 GitHub Environment 的主机地址、用户名、私钥等Secrets），在服务器上拉取最新 Points 镜像并运行容器更新服务。[[26]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L45-L48)[[27]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L46-L49)由于 Points 前端容器可能需要与 StoryApp 后端共存，部署时要选择合适的端口和路径。例如，可以将 Points 容器运行在服务器上的 5001 端口，并在服务器的 Nginx 配置中增加新站点或反向代理规则，将特定域名或路径转发到该容器。部署脚本可借鉴 StoryApp 的思路实现**无停机更新**：例如先拉取新镜像、启动新容器实例，再停止旧容器，实现滚动更新。[[28]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L32-L35)部署完毕后，可通过访问 Points 前端页面进行验证。
7. **测试与验证**：在整个迁移过程中分阶段进行验证。在容器化完成后，本地运行 docker build 和 docker run 验证 Points 前端容器能够正常提供页面。在 CI/CD 配置完成后，可尝试触发一次测试 workflow（比如推送一个测试分支触发 CI，或手动执行构建工作流）来验证流程是否成功。最后，在部署前可先在非生产环境试部署 Points（例如临时服务器或本地 compose up）进行验证，确保与StoryApp的机制兼容。

以上阶段划分确保将 StoryApp 成熟的 DevOps 机制逐步引入 Points 项目。在执行过程中，需要根据 Points 的具体情况进行调整，比如由于 Points 无后端代码，其 CI 流程会相对简化；又如部署时仅部署静态前端无需数据库迁移等。但总体而言，这些步骤将为 Points 项目建立起与 StoryApp 一致的容器化和持续集成/部署体系。

## 配置修改与文件调整建议

为顺利完成上述迁移，需要对 Points 仓库进行以下文件新增和配置修改：

* **新增 Dockerfile**（位于 Points 项目根目录）：采用 Node 18 构建前端、Nginx 容器提供静态资源的多阶段构建文件。确保在构建阶段安装 Points 前端依赖并执行 npm run build，然后在最终镜像中包含构建生成的静态文件（详见下文模板）。这样可将 Points 前端打包为可部署的Docker镜像，与 StoryApp 的镜像构建方式保持一致[[3]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L2-L10)。如果需要进一步优化镜像大小，可参考 StoryApp 做法使用 Alpine 基础镜像并清理不必要缓存。
* **新增 docker-compose.yml**（位于 Points 项目根目录）：用于定义 Points 容器服务的编排配置。包括：定义项目网络名（如 points-net），防止与其它应用网络冲突[[5]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L6-L14)；定义 app 服务，设置容器名称（如 points-app）、镜像名（指向构建的 Points 镜像或 GHCR 仓库镜像）以及端口映射等[[8]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L52-L60)。由于 Points 无数据库依赖，Compose 文件可精简为单一服务。可参考 StoryApp Compose 中 app 服务的环境变量用法，通过 env\_file: .env 统一加载配置[[7]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L44-L50)。确保端口映射不与 StoryApp 冲突，例如默认使用主机 5001 端口映射容器 80 端口（可通过变量 ${APP\_PORT} 调整）。如果需要日志持久化，可选择性地添加卷挂载（例如挂载容器内 /usr/share/nginx/html 以备调试，但一般静态内容不需要持久化）。
* **（可选）新增 docker-compose.dev.yml**：如果希望在容器中进行开发调试，可增加开发模式的 Compose 覆盖文件。该文件使用 Node 镜像运行 npm run dev 开发服务器，挂载源码以支持热更新[[29]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.dev.yml#L32-L40)。例如，可将 Points 前端源码挂载到 /app，命令设置为 npm install && npm run dev，并映射容器的 3000 端口用于本地开发预览。需要注意此容器主要供开发使用，不会上线部署。
* **新增 .env.example**：提供环境变量示例文件，列出 Points 项目可能用到的配置项，方便开发者复制为实际 .env 使用。考虑到 Points 前端可能未来与后端服务对接，可预留变量如 VITE\_API\_URL 等。目前可包含的项例如运行环境和端口：NODE\_ENV=production，APP\_PORT=5001 等。虽然前端应用敏感信息较少，但仍建议**不将配置硬编码**在前端代码中，而通过环境变量或构建参数传入，以便不同部署环境使用不同配置（如 API 地址、开启调试模式等）。同时，在 GitHub 仓库的 Settings 中添加相应的 Secrets（如部署服务器凭据、GHCR\_PAT 等），以备 CI/CD 工作流引用。[[19]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L39-L48)
* **新增 GitHub Actions 工作流文件**：在 .github/workflows/ 目录下添加前述 CI 和 CD 工作流配置文件，包括：
* ci.yml：Pull Request CI 流程，运行构建和检查。
* docker-build-push.yml：持续集成构建与推送镜像流程，在代码推送到主分支或打 tag 时执行，自动将 Points 镜像推送到 GHCR。[[24]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L291-L299)
* deploy-prod.yml：生产部署流程，手动触发，经由安全的 GitHub Environment 提供服务器连接凭据，实现一键部署更新 Points 容器。[[14]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L29-L37)

这些工作流需根据 Points 项目特点进行调整，例如 CI 中无需拆分前后端步骤，而只需构建前端；部署流程中镜像名称和容器名要改为 Points 相应值等。建议在 Action 中适当加入**缓存**机制以加速构建，比如缓存 npm 的依赖目录（StoryApp 工作流未明显使用缓存，可在 Points 工作流中新增此优化）。另外要注意引用正确的 Secrets（比如使用 ${{ secrets.GHCR\_PAT }} 登录 GHCR，${{ secrets.PROD\_SSH\_KEY }}用于 SSH 等），保证敏感信息不泄露。

* **文档更新**：相应地，更新 Points 项目的 README 或部署文档，说明新的容器化启动方式和 CI/CD 流程。例如，添加如何使用 Docker 启动 Points 的说明，如何配置 .env，以及 CI/CD 状态徽章等。这有助于团队其他成员了解并遵循新的工作流程。StoryApp 提供了详细的部署指南文档作为参考[[30]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L26-L35)[[31]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L272-L280)。

以上修改完成后，Points 项目将在结构上增添容器化和CI所需的文件和配置，其仓库布局将更类似 StoryApp。例如，将出现 Dockerfile、docker-compose.yml、.github/workflows 等目录和文件。接下来给出部分核心配置文件的模板示例。

## GitHub Actions 工作流模板（Points 项目）

以下是针对 Points 项目设计的 GitHub Actions 工作流模板。请根据实际仓库名称、分支名称进行调整，并在 GitHub 仓库中添加所需的 Secrets 和 Environment。

### CI 工作流（ci.yml）

name: CI Quality Check  
  
# 触发：所有 Pull Request 提交  
on:  
 pull\_request:  
 branches: [ main, master ]  
  
jobs:  
 build-and-test:  
 runs-on: ubuntu-latest  
 steps:  
 - name: 🛎️ 检出代码  
 uses: actions/checkout@v3  
  
 - name: 📦 安装依赖  
 uses: actions/setup-node@v3  
 with:  
 node-version: 18  
 - run: npm ci  
  
 - name: 🔨 构建前端  
 run: npm run build  
  
 # 如果有测试脚本，可以在此处执行  
 # - name: 🧪 运行测试  
 # run: npm test  
  
 - name: ✅ 构建成功  
 run: echo "Build finished successfully."

**说明**：上述 CI 工作流在每次 PR 时运行，使用 Node 18 安装依赖并构建 Points 前端项目。如需包括其他质量检查（例如 ESLint、StyleLint 或单元测试），可在构建步骤前后添加。由于 Points 目前主要是前端界面，构建本身可以视为基本的正确性验证。未来可扩展更多检查步骤。

### 镜像构建与推送工作流（docker-build-push.yml）

name: Build and Push Docker Image  
  
# 触发：推送到主分支 或 打标签  
on:  
 push:  
 branches: [ main, master ]  
 release:  
 types: [created]  
  
jobs:  
 docker-build:  
 runs-on: ubuntu-latest  
 steps:  
 - name: 🛎️ 检出代码  
 uses: actions/checkout@v3  
  
 - name: 🐳 登录 GHCR  
 env:  
 CR\_USERNAME: ${{ github.actor }}  
 CR\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}  
 run: echo "$CR\_TOKEN" | docker login ghcr.io -u "$CR\_USERNAME" --password-stdin  
  
 - name: 🔨 构建 Points 镜像  
 run: |  
 TAG\_SHA="sha-${GITHUB\_SHA::7}"  
 IMAGE\_NAME="ghcr.io/haizhouyuan/points"  
 docker build -t $IMAGE\_NAME:$TAG\_SHA -t $IMAGE\_NAME:sha-latest .  
  
 - name: 📦 推送镜像到 GHCR  
 run: |  
 IMAGE\_NAME="ghcr.io/haizhouyuan/points"  
 TAG\_SHA="sha-${GITHUB\_SHA::7}"  
 docker push $IMAGE\_NAME:$TAG\_SHA  
 docker push $IMAGE\_NAME:sha-latest  
  
 - name: 🔍 验证镜像运行  
 run: |  
 # 运行容器测试静态页面是否可访问  
 docker run -d --name points-test -p 5002:80 ghcr.io/haizhouyuan/points:sha-latest  
 sleep 5  
 curl -f http://localhost:5002 || (echo "Health check failed" && docker logs points-test && exit 1)  
 docker stop points-test && docker rm points-test  
  
 # （可选）漏洞扫描步骤：  
 # - name: 🚨 扫描镜像漏洞  
 # uses: aquasecurity/trivy-action@v0.11.1  
 # with:  
 # image-ref: ghcr.io/haizhouyuan/points:sha-latest  
 # severity: HIGH,CRITICAL  
 # exit-code: 1 # 若存在高危漏洞则使流程失败

**说明**：该工作流在代码合并主分支或发布 Release Tag 时触发。[[32]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L276-L284)它使用 Docker CLI 将 Points 项目打包为镜像并推送到 GitHub 容器注册表。我们使用了两个标签：sha-<短SHA>用于唯一标识每次构建，sha-latest始终指向最新构建[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318)。推送完成后，通过启动容器并使用 curl 请求首页，验证静态页面能否正常提供。[[25]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L300-L305)（由于 Points 无后端 API 健康检查，此处以首页返回状态作为验证。）最后提供了一个可选的镜像安全扫描步骤，建议启用以检测高危漏洞[[33]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L24-L28)。如发现严重漏洞，可以阻止不安全的镜像发布。请确保在 GHCR 中为仓库配置了合适的权限，GitHub 默认的 GITHUB\_TOKEN 可用于推送属于当前仓库命名空间的镜像。[[24]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L291-L299)

### 部署工作流（deploy-prod.yml）

name: Deploy to Production  
  
# 手动触发部署  
on:  
 workflow\_dispatch:  
 inputs:  
 tag:  
 description: "部署的镜像标签（默认 sha-latest）"  
 required: false  
 default: "sha-latest"  
  
jobs:  
 deploy:  
 # 指定在 GitHub Environment "production" 下运行，以访问部署Secrets  
 runs-on: ubuntu-latest  
 environment: production  
 steps:  
 - name: 🚀 部署到服务器  
 uses: appleboy/ssh-action@v0.1.6  
 with:  
 host: ${{ secrets.PROD\_HOST }}  
 username: ${{ secrets.PROD\_USER }}  
 key: ${{ secrets.PROD\_SSH\_KEY }}  
 script: |  
 # 登录 GHCR 仓库  
 echo "$GHCR\_PAT" | docker login ghcr.io -u haizhouyuan --password-stdin  
 # 拉取指定标签的 Points 镜像  
 docker pull ghcr.io/haizhouyuan/points:${{ github.event.inputs.tag }}  
 # 停止并移除旧容器  
 docker rm -f points-app || true  
 # 以新镜像启动容器（映射到主机5001端口）  
 docker run -d --name points-app --restart unless-stopped -p 5001:80 ghcr.io/haizhouyuan/points:${{ github.event.inputs.tag }}  
 # 等待容器启动并简单健康检查  
 sleep 5  
 if docker ps | grep -q "points-app"; then echo "✅ 容器已启动"; fi

**说明**：该工作流用于将更新后的 Points 容器部署到生产服务器。[[14]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L29-L37)触发时可指定镜像标签参数（默认为最新镜像）。在作业中，我们使用 **SSH Action** 直接在远程服务器上执行部署命令。首先使用保存于 Secrets 的 GHCR\_PAT 登录 GHCR，然后拉取待部署的 Points 镜像。接着停止并删除旧的容器实例，再使用新镜像启动容器，映射容器内80端口到服务器的5001端口运行。[[34]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L19-L27)这样做可以让 Points 前端通过服务器端口 5001 提供服务（可通过 Nginx 反向代理到公网域名）。部署完成后，输出简单状态检查信息。如需更严格的健康检查，可在容器内托管一个静态文件检查或在部署脚本中对 curl http://localhost:5001 执行检查。此工作流使用了 GitHub **Environment production** 来管理敏感信息，确保只有受信任的流程才能访问服务器凭据[[35]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L43-L50)。请事先在仓库设置中配置好 production 环境的 Secrets（包括 PROD\_HOST、PROD\_USER、PROD\_SSH\_KEY 和 GHCR\_PAT 等）。部署前也需要在目标服务器上配置好 Docker 及必要的网络/域名解析，确保 Points 容器能够被终端用户访问。(*注：如果 StoryApp 与 Points 部署在同一服务器且使用相同域名，不同应用需配置不同的访问路径或二级域名，并在服务器的 Nginx 上进行区分映射。*)

## Dockerfile 和 Docker Compose 模板（Points 项目）

下面提供 Points 项目的 Dockerfile 以及主要的 Docker Compose 配置模板。这些模板结合了 StoryApp 的容器化经验并针对 Points 的实际情况进行了简化。

### Dockerfile（Points）

# 基于 Node.js 18 构建阶段，安装依赖并编译前端  
FROM node:18-alpine AS builder  
WORKDIR /app  
  
# 将 package.json 和 lock 文件拷贝进来安装依赖  
COPY package.json package-lock.json ./  
RUN npm ci  
  
# 拷贝源代码并构建  
COPY . .  
RUN npm run build  
  
# 使用 Nginx 轻量级镜像作为运行阶段  
FROM nginx:1.23-alpine  
# 拷贝前端构建产物到 Nginx 默认静态目录  
COPY --from=builder /app/build /usr/share/nginx/html  
  
# （可选）设定非 root 用户运行 Nginx，提高安全性  
# RUN adduser -D -H -u 1001 points && \  
# sed -i 's/user nginx;/user points;/' /etc/nginx/nginx.conf  
  
# 暴露80端口（Nginx 默认监听端口）  
EXPOSE 80  
# 采用 Nginx 前台运行（镜像自带CMD，不需要显式指定）

**说明**：此 Dockerfile 分为两个阶段：第一阶段使用 Node 18 Alpine 镜像安装 Points 项目依赖并执行 npm run build 构建静态文件；第二阶段使用 Nginx Alpine 镜像作为运行容器，将前一阶段生成的前端静态文件拷贝至 Nginx 的默认网页目录。[[36]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L74-L82)这样构建的最终镜像只包含静态站点和 Nginx，体积小且无需包含源代码。Nginx 会在容器启动时自动服务这些静态文件。在该文件中，我们也示例了如何**降低权限**运行容器：可以添加一个非 root 用户并修改 Nginx 配置使其以该用户运行，从而增强安全性（上面的相关指令默认注释，可视需要启用）。由于 Points 是纯前端应用，不存在后端服务进程，故无需定义 ENTRYPOINT 或 CMD，沿用 Nginx 镜像内置的启动命令即可。构建完成的镜像可在本地通过 docker build -t points:local . 进行测试运行。

### docker-compose.yml（Points）

version: "3.8"  
name: points # 项目Compose命名空间  
networks:  
 points-net:  
 driver: bridge  
  
services:  
 app:  
 container\_name: points-app  
 # 镜像来源（默认使用GHCR镜像，可通过APP\_TAG指定特定tag）  
 image: ghcr.io/haizhouyuan/points:${APP\_TAG:-latest}  
 # 如需本地构建测试，可改用以下build配置  
 # build: .  
 env\_file:  
 - .env # 载入环境变量配置  
 environment:  
 - NODE\_ENV=production  
 ports:  
 - "${APP\_PORT:-5001}:80" # 将容器80端口映射到主机端口（默认5001）  
 networks:  
 - points-net  
 restart: unless-stopped  
 # 前端应用无特殊健康检查端点，可选配简单的HTTP检查  
 healthcheck:  
 test: ["CMD-SHELL", "wget -qO- http://localhost || exit 1"]  
 interval: 30s  
 timeout: 5s  
 retries: 3

**说明**：上述 Compose 配置定义了 Points 应用的单容器服务。我们命名服务为 app（与 StoryApp 保持一致的命名习惯），将其容器命名为 points-app。镜像默认为从 GHCR 拉取 haizhouyuan/points 仓库的镜像，并允许通过环境变量 APP\_TAG 来指定标签（默认使用 latest 或上文 push 时设定的 sha-latest 标签）。这样在需要部署特定版本时，可临时设置 APP\_TAG 环境变量再启动容器。[[15]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L6-L14)**网络**方面，创建了自有的 points-net bridge网络，保证与其他应用（如StoryApp）的网络隔离。[[5]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L6-L14)**端口**映射上，将容器内部的 80 端口映射到主机的 5001 端口（可在 .env 中通过 APP\_PORT 修改），避免与 StoryApp 后端使用的5000端口冲突。[[34]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L19-L27)由于 Points 没有数据库依赖，Compose 中不包含数据库服务，亦无需定义卷。唯一挂载的配置为 .env 文件，方便在容器启动时读取环境变量。上面配置还示范性地加入了一个简易**健康检查**，通过容器内执行 wget 请求本地网页判断服务可用性。一旦 Points 前端页面可被访问，则 healthcheck 通过；如 Nginx 进程异常退出则检查会失败，从而让 Compose/Swarm 捕捉到容器异常状态。健康检查并非必须，可根据需要保留或删除。

如果需要在开发环境使用 Compose 来运行 Points 应用进行调试，可增加一个 docker-compose.dev.yml 覆盖配置。例如：

# docker-compose.dev.yml  
services:  
 app:  
 image: node:18-alpine # 开发模式使用 Node 镜像直接运行开发服务器  
 working\_dir: /app  
 container\_name: points-dev  
 environment:  
 - NODE\_ENV=development  
 command: sh -c "npm install && npm run dev"  
 volumes:  
 - .:/app # 挂载源码以实现热更新  
 - /app/node\_modules # 匿名卷避免覆盖依赖  
 ports:  
 - "3000:3000" # 开发服务器端口

开发配置使我们可以通过 docker compose -f docker-compose.yml -f docker-compose.dev.yml up 来启动开发模式容器。[[29]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.dev.yml#L32-L40)在该模式下，Points 前端会在容器内运行 vite 开发服务器，监听 3000 端口并映射到宿主机，方便在浏览器中访问进行调试。代码改动会实时反映，无需每次手动构建镜像。注意，这一容器仅供本地开发使用，生产环境下应仍以前述正式镜像运行。

## 镜像命名规范及发布策略

为了与 StoryApp 保持一致并确保镜像管理清晰，Points 项目应遵循类似的**镜像命名约定**和发布策略：

* **命名规范**：采用 **${{owner}}/${{repo}}:标签** 的形式，其中标签包括：
* sha-<短提交哈希>：标识具体一次构建，对应某个 commit。[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318)
* sha-latest：始终指向最近一次构建的镜像，用于方便地在 Compose 或部署脚本中引用最新镜像。[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318)
* **版本标签**：当Points项目发布版本时，可使用语义化版本号作为标签（例如 v1.0.0），方便回滚和多环境部署。[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318)另外也可约定 latest 指向最新稳定版发行。

例如，Points 项目构建后在 GHCR 上的镜像名将形如：

ghcr.io/haizhouyuan/points:sha-latest  
ghcr.io/haizhouyuan/points:sha-7b3c1e2 # 具体commit  
ghcr.io/haizhouyuan/points:v1.0.0 # 发布版

* **推送策略**：利用 GitHub Actions 自动执行镜像构建和推送。[[13]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L21-L28)开发者不需要手动构建发布，但在本地调试镜像时，可使用以下命令：
* # 本地构建测试  
  docker build -t ghcr.io/haizhouyuan/points:test .  
  # 登录GHCR（如未配置免密）  
  echo $GHCR\_PAT | docker login ghcr.io -u <YourUsername> --password-stdin  
  # 推送测试镜像  
  docker push ghcr.io/haizhouyuan/points:test
* 建议在 CI 工作流完成镜像推送后，通过 GHCR 的 web 界面或 docker pull 命令验证镜像可正常拉取、启动。与此同时，确保 GHCR 仓库的**可见性**和**权限**设置正确：如果 Points 仓库为私有仓库，其对应的GHCR镜像默认为私有，需要在部署服务器上使用 ${{ secrets.GHCR\_PAT }} 登录后拉取；若希望镜像公开，则需将其设置为public，以便无需凭证即可拉取。
* **部署发布**：采用**滚动更新**策略发布新镜像。具体流程为：在 Points 有新功能合并到主分支后，CI 自动构建出新镜像（带有唯一标签）；运维人员通过部署工作流或脚本拉取新镜像，在服务器上启动新容器实例；经验证无误后，停止旧容器。这样在更新过程中旧版本仍然服务，直到新版本就绪，实现服务平滑过渡。[[28]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L32-L35)由于 Points 前端为无状态静态应用，这种更新相对简单，无需考虑会话或数据库迁移问题。
* **回滚方案**：如果新版本出现问题，由于我们保留了历史镜像（通过提交 SHA 或版本标签），可快速回滚：只需将部署脚本中的镜像标签改为上一个稳定版本，并重新部署容器即可。[[37]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docs/DEPLOYMENT_WORKFLOW.md#L198-L206)建议在实际部署前对新版本前端做充分测试（例如在本地或预备环境运行 docker compose up 进行验证），将故障风险降到最低。

## 安全性与可维护性考量

在迁移过程中，需要关注**安全**和**可维护性**方面的细节，确保 Points 项目在引入容器化和CI/CD后依然保持健壮、安全：

* **敏感信息管理**：延续 StoryApp 的做法，将任何敏感配置放入环境变量/Secrets 中管理，而非硬编码在仓库。例如，如果未来 Points 引入了第三方 API 调用（如积分数据后端服务的 API\_KEY），应将其添加到 .env 文件和 GitHub Secrets，并在前端构建时注入（通过 Vite 的 import.meta.env 机制）。目前 Points 无需存储密钥，但部署相关的凭据（SSH 私钥、GHCR PAT 等）必须保存在 GitHub Secrets/Environment 中，绝不可直接写在工作流文件里。[[19]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L39-L48)
* **容器最小权限**：参考 StoryApp 容器安全实践，不以 root 身份运行应用。[[17]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L55-L63)Points 前端容器采用 Nginx，其官方镜像默认工作进程以 nginx 用户运行（虽然主进程仍需 root 来绑定80端口）。可以考虑进一步使用无特权版 Nginx 镜像或手动降权运行（正如 Dockerfile 中示例的那样），以减少潜在风险。与此同时，容器只暴露必要的端口（Points 容器只需暴露 80）[[18]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L24-L31)；对于不需要对外开放的服务（本例中无此类服务），在 Compose 中应避免映射端口。总之，遵循最小权限原则配置容器。
* **分离环境**：确保开发、测试、生产三套环境隔离运行，避免互相影响。StoryApp 通过不同的 Compose 覆盖文件和端口规划做到了这一点[[38]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L14-L22)。Points 项目虽然目前规模较小，也应有类似意识。例如，本地开发时使用 3000 端口、GHCR 验证时使用 5002 端口、生产部署使用 5001/80 端口，互不冲突。[[39]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L12-L20)不同环境可采用各自的配置文件（如 .env.development 与 .env.production）管理特定参数，从而减少人为错误。
* **依赖与镜像更新**：定期更新基础镜像和前端依赖，保持安全。StoryApp CI 中提到可使用 Renovate/Dependabot 自动更新依赖[[40]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L92-L98)，Points 项目也可引入这些工具，及时获取安全补丁。在镜像构建过程中加入漏洞扫描（如上文中的 Trivy 步骤）可以及早发现安全隐患。[[33]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L24-L28)另外，谨慎评估 Points 前端所使用的第三方库，移植CI后可以开启构建产物的体积分析或依赖审计，提升可维护性。
* **监控与日志**：虽然 Points 前端没有后端日志，但部署后仍应监控容器的运行状态。例如，通过 Docker 自带的日志 (docker logs points-app) 可以观察到 Nginx 的访问日志和错误日志。如需要，考虑在 Nginx 配置中调优日志级别，并将日志输出路径挂载主机以便长期保存分析（默认Nginx日志在容器内 /var/log/nginx/）。Compose 已加入基础的健康检查，可以及早发现容器故障。[[20]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L61-L67)对于前端性能和可用性，也可以借助前端监控工具（如果以后集成）来提升可维护性。
* **权限分离与代码审查**：启用 CI 工作流后，可在仓库设置中实施分支保护，要求所有 Pull Request 必须通过CI检查才能合并[[41]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L96-L98)。这保证了进入主分支的代码经过基本验证，减少引入破坏性变更的可能。此外，通过 Claude 等代码审查工具（StoryApp 已经尝试）可以提前发现安全漏洞或代码异味[[23]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L16-L19)，Points 项目后续也可以考虑引入，将安全左移到开发阶段。

综上，按照上述迁移方案，Points 项目将获得与 StoryApp 类似的容器化部署和 CI/CD 能力。在迁移过程中需充分测试各环节的适配性。例如，构建出的 Docker 镜像是否正确提供前端页面，工作流中的环境变量是否匹配 Points 项目需求等。通过一步步严格对照 StoryApp 的经验进行调整，最终 Points 将具备**容器化运行**（方便部署到各类服务器或云环境）、**持续集成**（每次改动自动验证）、**持续部署**（一键发布更新）的能力。同时在安全和运维上也达到生产要求，从而为 Points 项目的后续迭代提供稳定、高效的基础设施支持。[[42]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L4-L8)

[[1]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/README.md#L14-L19) README.md

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/README.md>

[[2]](https://github.com/haizhouyuan/points/blob/69a6ac25bc55e4b40ca406de323d74172ff666f3/README.md#L2-L5) README.md

<https://github.com/haizhouyuan/points/blob/69a6ac25bc55e4b40ca406de323d74172ff666f3/README.md>

[[3]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L2-L10) [[17]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L55-L63) [[36]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile#L74-L82) Dockerfile

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/Dockerfile>

[[4]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L22-L30) [[5]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L6-L14) [[6]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L11-L19) [[7]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L44-L50) [[8]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L52-L60) [[9]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L26-L34) [[20]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml#L61-L67) docker-compose.yml

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.yml>

[[10]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/.env.example#L4-L12) [[11]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/.env.example#L20-L28) .env.example

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/.env.example>

[[12]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L12-L20) [[13]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L21-L28) [[14]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L29-L37) [[19]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L39-L48) [[23]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L16-L19) [[26]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L45-L48) [[27]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L46-L49) [[28]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L32-L35) [[33]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L24-L28) [[35]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L43-L50) [[40]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L92-L98) [[41]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L96-L98) [[42]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md#L4-L8) CICD\_SETUP.md

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/CICD_SETUP.md>

[[15]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L6-L14) [[18]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L24-L31) [[34]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml#L19-L27) docker-compose.ghcr.yml

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.ghcr.yml>

[[16]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L310-L318) [[24]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L291-L299) [[25]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L300-L305) [[30]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L26-L35) [[31]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L272-L280) [[32]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L276-L284) [[38]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L14-L22) [[39]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md#L12-L20) DOCKER\_DEPLOYMENT\_GUIDE.md

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/DOCKER_DEPLOYMENT_GUIDE.md>

[[21]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.dev.yml#L44-L52) [[29]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.dev.yml#L32-L40) docker-compose.dev.yml

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docker-compose.dev.yml>

[[22]](https://github.com/haizhouyuan/points/blob/69a6ac25bc55e4b40ca406de323d74172ff666f3/vite.config.ts#L52-L60) vite.config.ts

<https://github.com/haizhouyuan/points/blob/69a6ac25bc55e4b40ca406de323d74172ff666f3/vite.config.ts>

[[37]](https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docs/DEPLOYMENT_WORKFLOW.md#L198-L206) DEPLOYMENT\_WORKFLOW.md

<https://github.com/haizhouyuan/storyapp/blob/e16cdf2f2323484768588409987eeb6870302c2a/docs/DEPLOYMENT_WORKFLOW.md>