# Internship Documentation

*Documentation of tasks performed during my internship.*

## Overview

This document outlines the tasks completed and challenges faced during my internship, focusing on containerization, cloud deployment, CI/CD pipelines, and feature management. Each day’s work is summarized with key activities and difficulties.

## Summary of Tasks

1. **Day 1: Local Strapi Deployment** – Set up Strapi CMS locally with Docker and native Node.js.
2. **Day 2: Containerize Strapi** – Containerized Strapi with custom Dockerfile and docker-compose.
3. **Day 3: Strapi + PostgreSQL + Nginx Setup** – Dockerized Strapi with PostgreSQL and Nginx reverse proxy.
4. **Day 6: Deploy Strapi on AWS EC2** – Deployed Strapi on EC2 using Terraform and Bash.
5. **Day 7: Strapi CI/CD Deployment** – Automated EC2 deployment with GitHub Actions and Terraform.
6. **Day 9: Strapi on AWS ECS Fargate** – Deployed Strapi on ECS Fargate with Terraform.
7. **Day 10: Automate Strapi on ECS Fargate** – Automated ECS deployment with GitHub Actions.
8. **Day 11: Strapi on ECS with CloudWatch** – Added CloudWatch monitoring for ECS deployment.
9. **Day 14: Strapi on ECS Fargate Spot** – Deployed Strapi using FARGATE\_SPOT.
10. **Day 15: Strapi Content and API Access** – Configured and tested Strapi public APIs.
11. **Day 16: Blue/Green Deployment** – Implemented Blue/Green deployment with CodeDeploy.
12. **Day 20: Automate Blue/Green Deployment** – Automated Blue/Green deployment with GitHub Actions.
13. **Day 22: Docker Swarm and Cronjobs** – Implemented Docker Swarm with NGINX and cron jobs.
14. **Day 23: Unleash Feature Flags with React** – Integrated Unleash for feature toggling in React app.
15. **Day 24: Internship Tasks Documentation** – Documented all tasks and difficulties from Day 1 to Day 24.

## Day 1: Local Strapi Deployment

* **Tasks**:
  + Set up Strapi CMS locally using Docker and documented native setup.
  + Cloned GitHub repository, used docker-compose to start container.
  + Initialized Strapi project with npx create-strapi@latest.
  + Configured MySQL (host, port, credentials) instead of SQLite.
  + Enabled example data, TypeScript, and initialized Git repository.
  + Started development server, accessed admin panel at http://localhost:1337/admin.
  + Documented native Node.js and SQL database setup.
* **Difficulties**: None

## Day 2: Containerize Strapi

* **Tasks**:
  + Containerized Strapi project using custom Dockerfile (Node.js 22 Alpine).
  + Configured environment variables for SQL database (client, host, port, credentials).
  + Defined working directory, copied files, exposed port 1337.
  + Demonstrated deployment via:
    - docker run: Built custom image, passed environment variables.
    - docker-compose: Spun up Strapi and MySQL containers.
  + Verified admin panel access at http://localhost:1337/admin.
* **Difficulties**: None

## Day 3: Strapi + PostgreSQL + Nginx Setup

* **Tasks**:
  + Set up Dockerized Strapi with PostgreSQL and Nginx.
  + Created user-defined Docker network for container communication.
  + Provisioned PostgreSQL container with volume for persistent storage.
  + Connected Strapi container to PostgreSQL via environment variables.
  + Configured Nginx reverse proxy, forwarding port 80 to Strapi’s 1337.
  + Verified admin panel at http://localhost:80/admin.
  + Implemented setup with docker-compose for simplified orchestration.
  + Ensured Strapi security by exposing only Nginx to host.
* **Difficulties**: None

## Day 4 & 5: Weekend

* No tasks.

## Day 6: Deploy Strapi on AWS EC2

* **Tasks**:
  + Deployed Dockerized Strapi on AWS EC2 using Terraform and Bash script.
  + Used custom Strapi Docker image with PostgreSQL environment variables.
  + Configured AWS credentials and Terraform project.
  + Customized main.tf (AMI, instance type, VPC, security group, SSH key).
  + Deployed EC2 instance, ran strapi-deployment.sh via user\_data.
  + Accessed admin panel at http://<EC2-IP>:1337/admin.
* **Difficulties**:
  + Docker containers failed to start (Docker not pre-installed in AMI). Fixed by updating Bash script to include Docker installation.

## Day 7: Strapi CI/CD with GitHub Actions

* **Tasks**:
  + Automated Strapi deployment on EC2 using GitHub Actions and Terraform.
  + Set up GitHub Secrets for AWS and DockerHub credentials.
  + Configured workflows:
    - ci.yml: Build and push Docker image to DockerHub.
    - terraform.yml: Provision EC2 and deploy app.
  + Verified admin panel at http://<public-ip>:1337/admin.
* **Difficulties**:
  + Terraform pipeline failed due to missing image tag in CI pipeline. Fixed by writing image tag to file, committing, and reading in Terraform pipeline.

## Day 8: No Task

* No tasks assigned.

## Day 9: Strapi on AWS ECS Fargate

* **Tasks**:
  + Deployed Strapi on AWS ECS Fargate using Terraform.
  + Created main.tf, ecr.tf, variables.tf for ECS service, task definition, IAM roles, ECR.
  + Ran Terraform commands to provision infrastructure.
  + Launched Strapi container behind Application Load Balancer (ALB).
  + Accessed admin panel at http://<app\_lb\_dns>/admin.
* **Difficulties**:
  + ECS tasks failed to pull Docker image. Fixed by correcting Security Group rules.
  + Strapi rejected ALB connections. Fixed by editing Strapi config to accept all hosts.

## Day 10: Automate Strapi on ECS Fargate

* **Tasks**:
  + Automated Strapi deployment on ECS Fargate using GitHub Actions and Terraform.
  + Configured GitHub Secrets for AWS credentials.
  + Set up workflows:
    - ci.yml: Build and push Docker image to ECR on push to main.
    - terraform.yml: Manually triggered to deploy infrastructure.
  + Customized Terraform files for ECS and AWS resources.
  + Verified admin panel at http://<app\_lb\_dns>/admin.
* **Difficulties**: None

## Day 11: Strapi on ECS with CloudWatch

* **Tasks**:
  + Automated Strapi deployment on ECS Fargate using GitHub Actions and Terraform.
  + Configured workflows:
    - ci.yml: Build and push Docker image to ECR.
    - terraform.yml: Manually triggered to deploy infrastructure.
  + Integrated CloudWatch Logs, Alarms, and Dashboard for ECS monitoring.
  + Set up log groups and alarms for CPU/memory usage.
  + Created CloudWatch Dashboard for service metrics.
  + Verified admin panel at ALB DNS.
* **Difficulties**:
  + Struggled with configuring CloudWatch dashboard widget metric sources.

## Day 12 & 13: Weekend

* No tasks.

## Day 14: Strapi on ECS Fargate Spot

* **Tasks**:
  + Deployed Strapi on AWS ECS using FARGATE\_SPOT launch type.
  + Configured Terraform files for infrastructure.
  + Initialized and applied Terraform plan for ECS services with Spot capacity.
  + Verified admin panel at ALB DNS.
* **Difficulties**: None

## Day 15: Strapi Content and API Access

* **Tasks**:
  + Deployed Strapi on ECS Fargate and verified via ALB.
  + Created content for collections: Article, Author, About, Category.
  + Configured public API permissions for find and findOne.
  + Tested API endpoints with curl, retrieved JSON responses.
* **Difficulties**:
  + Permission denied errors on public APIs due to missing Strapi role config.
  + Content not visible due to unpublished entries. Fixed by publishing content.

## Day 16: Blue/Green Deployment with CodeDeploy

* **Tasks**:
  + Implemented Blue/Green deployment for Strapi on ECS Fargate using AWS CodeDeploy and Terraform.
  + Verified infrastructure: ECS Cluster, ALB, listener rules, CodeDeploy setup.
  + Initiated manual deployment via CodeDeploy console with valid appspec.yaml.
  + Ensured correct TaskDefinition and port mapping (1337).
* **Difficulties**:
  + Missed IAM permissions for CodeDeploy. Fixed by updating role policy.
  + Required careful ALB configuration for blue/green deployment.

## Day 17: No Task

* No tasks assigned.

## Day 18 & 19: Weekend

* No tasks.

## Day 20: Automate Blue/Green Deployment

* **Tasks**:
  + Automated Strapi deployment on ECS Fargate with Blue/Green strategy using GitHub Actions and CodeDeploy.
  + Configured workflows:
    - CI.yml: Build, tag (Git commit hash), push Docker image to ECR.
    - TF\_Deploy.yml: Provision infrastructure via Terraform.
    - CODE\_DEPLOY.yml: Trigger Blue/Green deployment.
    - Terraform Destroy: Manually tear down infrastructure.
  + Set up GitHub Secrets for AWS credentials.
  + Configured appspec.json for CodeDeploy.
* **Difficulties**:
  + CODE\_DEPLOY.yml failed due to appspec.yaml parsing issue. Switched to appspec.json and used jq tool to fix.

## Day 21: No Task

* No tasks assigned.

## Day 22: Docker Swarm and Cronjobs

* **Tasks**:
  + Learned and implemented Docker Swarm for container orchestration.
  + Provisioned 3 EC2 instances as Swarm nodes using Terraform.
  + Initialized Swarm Manager, joined Worker nodes with docker swarm join.
  + Deployed NGINX service with 3 replicas using docker service create.
  + Implemented cron jobs in Swarm using crazy-max/swarm-cronjob.
* **Difficulties**:
  + Network issues due to blocked ports (2377, 7946, 4789). Fixed by updating Security Group rules.
  + Challenges understanding and setting up swarm-cronjob due to lack of native cron support.

## Day 23: Unleash Feature Flags with React

* **Tasks**:
  + Integrated Unleash feature flag system into React app for dynamic toggling.
  + Set up Docker Compose for:
    - unleash-db (PostgreSQL).
    - unleash (Unleash Server).
    - proxy (Unleash Proxy).
  + Created showLiveClock toggle in Unleash dashboard.
  + Initialized Unleash client in main.jsx with @unleash/proxy-client-react.
  + Used useFlag() hook in App.jsx to toggle live clock component.
  + Verified toggling via Unleash UI without redeployment.
* **Difficulties**:
  + App failed to render due to incorrect Unleash client setup. Fixed by wrapping App with FlagProvider.
  + Confusion between admin and frontend tokens caused proxy failure. Fixed by using correct client token.

## Day 24: Documentation

* **Tasks**:
  + Documented all tasks and difficulties from Day 1 to Day 24.
* **Difficulties**: None

## Key Achievements

* Mastered containerization with Docker and Docker Compose.
* Gained expertise in AWS services (EC2, ECS Fargate, ALB, CodeDeploy, CloudWatch).
* Automated deployments using GitHub Actions and Terraform.
* Implemented advanced deployment strategies like Blue/Green.
* Integrated feature flag management with Unleash in a React app.
* Overcame technical challenges through debugging and configuration adjustments.