# **DevOps Research Task – Infrastructure Setup & CLI Flow for Backend.im**

**Author:** Yemisi Okunrounmu  
 **Role:** DevOps Intern

## **1. Objective**

Design a simple, cost-efficient infrastructure and workflow that allows a developer to push and deploy backend code to **Backend.im** using the **Claude Code CLI** and open-source tools, with minimal manual configuration.

The goal: **one-command deployment** from local development to live environment, ensuring consistency, automation, and minimal friction.

## **2. Understanding Backend.im**

* **.im** is the **top-level domain** of the service, similar to .com or .io.
* Backend.im is the **platform/service** where your backend code runs.
* The developer interacts with Backend.im **via CLI commands**, not manually logging into the platform.
* All deployment steps (code push, build, container deployment) are handled automatically.

## **3. Proposed Architecture**

Developer Laptop

│

│ (1) Code push / CLI command

▼

Version Control (GitHub/GitLab)

│

│ (2) CI/CD pipeline triggered

▼

CI/CD Pipeline (GitHub Actions / Jenkins)

│

│ (3) Build, test, package container

▼

Container Registry (Docker Hub / GitHub Packages)

│

│ (4) CLI triggers Backend.im deployment

▼

Backend.im Environment (Staging/Production)

│

│ (5) Automated service start & health check

▼

Live Application Accessible to Users

## **4. CLI Interaction with Backend.im**

1. **Local Development**
   * Developer writes code locally and tests it.
2. **Push/Deploy via CLI**

Example command:  
  
 claude deploy --target backend.im --env staging

* + CLI handles:  
    - Pushing code to GitHub
    - Triggering CI/CD workflow

1. **CI/CD Pipeline**
   * Runs unit tests
   * Builds Docker image or deployable artifact
   * Pushes image to container registry
2. **Backend.im Deployment**
   * CLI instructs Backend.im to pull the latest image
   * Backend.im sets environment variables, secrets, and runs the container
   * Health checks automatically verify service is live
3. **Result**
   * Service is deployed and accessible at https://backend.im/<service>
   * Developer does **not handle deployment manually**
   * Rollbacks or redeploys can be triggered via CLI if needed

## **5. Tools & Frameworks**

|  |  |  |
| --- | --- | --- |
| **Tool/Framework** | **Purpose** | **Reason** |
| Git / GitHub | Version control | Open-source, widely used |
| Claude Code CLI | Deployment trigger | Simplifies developer workflow |
| Docker / Docker Hub | Containerization | Standardized runtime and distribution |
| GitHub Actions / Jenkins | CI/CD automation | Free/open-source pipelines |
| HashiCorp Vault | Secret management | Secure credentials, minimal config |
| Backend.im | Hosting & deployment | Endpoint for live app |
| Nginx | Reverse proxy/load balancing | Optional for multi-service apps |

## **6. Local Setup Flow**

1. Developer clones repo or edits code locally.

Developer runs CLI command, e.g.:  
  
 claude deploy --target backend.im --env staging

1. CLI pushes code to GitHub → triggers CI/CD pipeline → builds image → pushes to registry.
2. Backend.im automatically pulls the image → deploys → health checks run.
3. CLI confirms the service is live.

## **7. High-Level Deployment Sequence Diagram (Textual)**

[Developer Laptop]

│ "claude deploy"

▼

[GitHub Repo]

│ trigger CI/CD

▼

[CI/CD Pipeline]

│ build/test/push

▼

[Container Registry]

│ image pull

▼

[Backend.im Environment]

│ start containers & run health checks

▼

[Live Application]

* .im represents the platform domain where the service is deployed.
* CLI abstracts all deployment steps, enabling **one-command deployment**.

## **8. Minimal Custom Code**

* CLI command configuration (environment, release ID)
* CI/CD pipeline YAML for automation
* Optional health-check scripts

All other operations rely on open-source tools and prebuilt Docker images.

## **9. Benefits of This Setup**

* **One-command deployment** for developers
* **Open-source and cost-efficient** tools
* **Minimal manual configuration**
* **Automatic verification** and rollback
* **Scalable for multiple services/environments**

**Testing external access:** We optionally used ngrok to expose the local Nginx endpoint (http://localhost:8080) to a public URL (https://jarrett-semisuccess-ungladly.ngrok-free.dev). This allowed external verification of the Blue/Green deployment without exposing the local machine directly.