CI/CD for Any Project — Kid-Simple Guide (GitHub Actions → AWS EKS)

> Think of this like delivering pizza: **Code** = dough, **Docker** = box, **ECR** = pizza shelf, **EKS** = delivery team, **Ingress/ALB** = your house gate, **Route53** = the address board. 🍕

---

0) What You'll Build (The Flow)

1. You push code to **GitHub**.
2. **GitHub Actions** runner wakes up and builds a **Docker image**.
3. Image is pushed to **Amazon ECR** (image storage).
4. We tell **Amazon EKS** (Kubernetes) to run that image.
5. An **ALB** (load balancer) gets a hostname.
6. **Route 53** creates a DNS record like `your-app.apps-aws.com` pointing to the ALB.

```
Dev writes code → GitHub → Runner → Docker build → ECR → EKS Deployment → ALB →
Route53 DNS
```

---

1) Prerequisites (Checklist)

- AWS account + IAM permissions to use ECR, EKS, IAM, Route53, Secrets Manager
- EKS cluster created (e.g., `dev-eks-2`) and `kubectl` access works
- Domain & hosted zone (e.g., `apps-aws.com`)
- Self-hosted **GitHub Actions Runner** on EC2 (or use GitHub-hosted with AWS OIDC)
- Docker & AWS CLI installed where runner executes

> Tip: On self-hosted EC2, prefer an **Instance Profile** (IAM role on EC2) instead of static AWS keys.

---

2) Repository Structure (Template)

```
.
├─ app/                   # your application code
├─ Dockerfile             # how to build the image
├─ .env.example           # sample local env file (never commit real secrets)
├─ cicd/
│  ├─ project.yaml        # project config (names, ports, DNS, etc.)
│  └─ templates/
│      ├─ deployment.yaml.tpl
│      ├─ service.yaml.tpl
│      └─ ingress.yaml.tpl
├─ terraform/            # IAM roles, IRSA, policies (optional if infra is pre-
created)
└─ .github/
   └─ workflows/
      └─ deploy.yml      # GitHub Actions workflow
```

---

3) Fill Project Config (One File, Many Projects)

`cicd/project.yaml`

```yaml
app_name: myapp
namespace: cicd-myapp
cluster_name: dev-eks-2
region: us-east-1
account_id: "123456789012"
ecr_repo: cicd-microservices
container_port: 5000
service_port: 80
ingress:
  host: myapp.apps-aws.com
  class: alb
  certificate_arn: arn:aws:acm:us-east-1:123456789012:certificate/xxxx
  subnets: [subnet-aaaa, subnet-bbbb]
  security_groups: [sg-zzzz]
  healthcheck_path: /
```

Add/adjust fields as needed. Each new project gets its own `project.yaml`.

---

4) Dockerfile (Generic)

`Dockerfile`

```dockerfile
FROM python:3.11-slim
WORKDIR /app
COPY app/ /app
COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
```

```
EXPOSE 5000
CMD ["python", "main.py"]
```

> Node/Java? Use your language’s base image and build commands.

---

5) Kubernetes Templates (Generic)

5.1 Deployment template

`cicd/templates/deployment.yaml.tpl`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ${APP_NAME}-deployment
  namespace: ${NAMESPACE}
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ${APP_NAME}
  template:
    metadata:
      labels:
        app: ${APP_NAME}
    spec:
      serviceAccountName: ${SERVICE_ACCOUNT:-mta-service-account}
      containers:
      - name: ${APP_NAME}-container
        image: ${ACCOUNT_ID}.dkr.ecr.${REGION}.amazonaws.com/${ECR_REPO}:$
{IMAGE_TAG}
        imagePullPolicy: Always
        ports:
        - containerPort: ${CONTAINER_PORT}
        envFrom:
        - secretRef:
            name: app-secrets
```

5.2 Service template

`cicd/templates/service.yaml.tpl`

```
apiVersion: v1
kind: Service
metadata:
  name: ${APP_NAME}-service
  namespace: ${NAMESPACE}
spec:
  selector:
    app: ${APP_NAME}
  ports:
  - protocol: TCP
    port: ${SERVICE_PORT}
    targetPort: ${CONTAINER_PORT}
  type: ClusterIP
```

5.3 Ingress template (ALB)

`cicd/templates/ingress.yaml.tpl`

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ${APP_NAME}-ingress
  namespace: ${NAMESPACE}
  annotations:
    kubernetes.io/ingress.class: "${INGRESS_CLASS}"
    alb.ingress.kubernetes.io/scheme: "internet-facing"
    alb.ingress.kubernetes.io/certificate-arn: "${CERT_ARN}"
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80}]'
    alb.ingress.kubernetes.io/target-type: "ip"
    alb.ingress.kubernetes.io/group.name: "${APP_NAME}-group"
    alb.ingress.kubernetes.io/load-balancer-attributes:
"idle_timeout.timeout_seconds=60"
    alb.ingress.kubernetes.io/subnets: "${SUBNETS}"
    alb.ingress.kubernetes.io/security-groups: "${SECURITY_GROUPS}"
    alb.ingress.kubernetes.io/healthcheck-path: "${HEALTHCHECK_PATH}"
spec:
  rules:
  - host: ${HOST}
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ${APP_NAME}-service
            port:
              number: ${SERVICE_PORT}
```

Separator tip: Between multiple YAML resources use `---`, not `___`.

6) Secrets (Don't hardcode!)

- Keep **.env** only for local dev; don't commit real values.
- Store real secrets in **AWS Secrets Manager**; mount into pods using:
- IRSA + external-secrets (recommended), or
- Kubernetes `Secret` created at deploy time from GitHub Actions.

Example (simple K8s Secret definition)

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
  namespace: ${NAMESPACE}
type: Opaque
stringData:
  DATABASE_URL: ${DATABASE_URL}
  API_KEY: ${API_KEY}
```

If a value contains `*` or special chars, quote it: `"p@ss*word"`.

7) Terraform (Infra/IAM/IRSA)

- Create IAM role for service account (IRSA) with trust policy for your EKS OIDC provider.
- Policy: allow `secretsmanager:GetSecretValue` (scoped to your secrets).
- Annotate Kubernetes ServiceAccount with the role ARN.
- Optional but great: store Terraform state in **S3** with **DynamoDB** locking.

You already have this mostly parameterized—just keep variables in `terraform/variables.tf` and `terraform/terraform.tfvars`.

8) GitHub Actions Workflow (Generic)

`.github/workflows/deploy.yml`

```
name: Deploy
on:
  push:
    branches: [ main ]

jobs:
  deploy:
    runs-on: [self-hosted, linux, x64]  # ensure your runner has these labels
    env:
      PROJECT_CONFIG: cicd/project.yaml

    steps:
    - uses: actions/checkout@v4

    - name: Install tools
      run: |
        sudo yum install -y jq || true
        sudo curl -L https://github.com/mikefarah/yq/releases/download/v4.44.3/
yq_linux_amd64 -o /usr/local/bin/yq
        sudo chmod +x /usr/local/bin/yq

    - id: read
      name: Read project config
      run: |
        REGION=$(yq '.region' $PROJECT_CONFIG)
        CLUSTER=$(yq '.cluster_name' $PROJECT_CONFIG)
        NAMESPACE=$(yq '.namespace' $PROJECT_CONFIG)
        APP=$(yq '.app_name' $PROJECT_CONFIG)
        ACCOUNT=$(yq '.account_id' $PROJECT_CONFIG)
        REPO=$(yq '.ecr_repo' $PROJECT_CONFIG)
        CPORT=$(yq '.container_port' $PROJECT_CONFIG)
        SPORT=$(yq '.service_port' $PROJECT_CONFIG)
        HOST=$(yq '.ingress.host' $PROJECT_CONFIG)
        CERT=$(yq '.ingress.certificate_arn' $PROJECT_CONFIG)
        SUBNETS=$(yq '.ingress.subnets | join(",")' $PROJECT_CONFIG)
        SGRPS=$(yq '.ingress.security_groups | join(",")' $PROJECT_CONFIG)
        echo "region=$REGION" >> $GITHUB_OUTPUT
        echo "cluster=$CLUSTER" >> $GITHUB_OUTPUT
        echo "namespace=$NAMESPACE" >> $GITHUB_OUTPUT
        echo "app=$APP" >> $GITHUB_OUTPUT
        echo "account=$ACCOUNT" >> $GITHUB_OUTPUT
        echo "repo=$REPO" >> $GITHUB_OUTPUT
        echo "cport=$CPORT" >> $GITHUB_OUTPUT
        echo "sport=$SPORT" >> $GITHUB_OUTPUT
        echo "host=$HOST" >> $GITHUB_OUTPUT
        echo "cert=$CERT" >> $GITHUB_OUTPUT
        echo "subnets=$SUBNETS" >> $GITHUB_OUTPUT
        echo "sgrps=$SGRPS" >> $GITHUB_OUTPUT

    - name: ECR login
      run: |
        aws ecr get-login-password --region ${{ steps.read.outputs.region }} \
          | docker login --username AWS --password-stdin \
            ${{ steps.read.outputs.account }}.dkr.ecr.$
{{ steps.read.outputs.region }}.amazonaws.com

    - name: Build & Push
      env:
        IMAGE_TAG: ${{ github.sha }}
      run: |
        REPO="${{ steps.read.outputs.account }}.dkr.ecr.$
{{ steps.read.outputs.region }}.amazonaws.com/${{ steps.read.outputs.repo }}"
        docker build -t "$REPO:$IMAGE_TAG" .
        docker push "$REPO:$IMAGE_TAG"
        echo "IMAGE_TAG=$IMAGE_TAG" >> $GITHUB_ENV

    - name: Update kubeconfig
      run: |
        aws eks update-kubeconfig --name ${{ steps.read.outputs.cluster }} --
region ${{ steps.read.outputs.region }}

    - name: Render manifests
      run: |
        export APP_NAME=${{ steps.read.outputs.app }}
        export NAMESPACE=${{ steps.read.outputs.namespace }}
        export ACCOUNT_ID=${{ steps.read.outputs.account }}
        export REGION=${{ steps.read.outputs.region }}
        export ECR_REPO=${{ steps.read.outputs.repo }}
        export CONTAINER_PORT=${{ steps.read.outputs.cport }}
        export SERVICE_PORT=${{ steps.read.outputs.sport }}
        export HOST=${{ steps.read.outputs.host }}
        export CERT_ARN=${{ steps.read.outputs.cert }}
        export INGRESS_CLASS=alb
        export SUBNETS=${{ steps.read.outputs.subnets }}
        export SECURITY_GROUPS=${{ steps.read.outputs.sgrps }}
        export HEALTHCHECK_PATH=/
        export IMAGE_TAG=${IMAGE_TAG}
        mkdir -p k8s
        envsubst < cicd/templates/deployment.yaml.tpl > k8s/deployment.yaml
        envsubst < cicd/templates/service.yaml.tpl    > k8s/service.yaml
        envsubst < cicd/templates/ingress.yaml.tpl    > k8s/ingress.yaml

    - name: Apply to cluster
      run: |
        kubectl get ns ${{ steps.read.outputs.namespace }} >/dev/null 2>&1 ||
kubectl create ns ${{ steps.read.outputs.namespace }}
        kubectl -n ${{ steps.read.outputs.namespace }} apply -f k8s/

    - name: Wait for rollout
      run: |
        kubectl -n ${{ steps.read.outputs.namespace }} rollout status deploy/$
{{ steps.read.outputs.app }}-deployment --timeout=300s

    - name: Create/Update DNS
      run: |
        INGRESS=${{ steps.read.outputs.app }}-ingress
        for i in {1..30}; do
          H=$(kubectl -n ${{ steps.read.outputs.namespace }} get ingress
$INGRESS -o jsonpath='{.status.loadBalancer.ingress[0].hostname}' 2>/dev/null)
          [ -n "$H" ] && break
          echo "Waiting for ALB hostname ($i/30)..."; sleep 15
        done
        [ -n "$H" ] || { echo "ERROR: No ALB hostname"; exit 1; }
        ZONE=$(echo ${{ steps.read.outputs.host }} | sed 's/^[^.]*\.//')
        HZID=$(aws route53 list-hosted-zones-by-name --dns-name "$ZONE" --query
"HostedZones[0].Id" --output text | sed 's|/hostedzone/||')
        cat > dns.json <<EOF
        {"Comment":"UPSERT","Changes":[{"Action":"UPSERT","ResourceRecordSet":
{"Name":"${{ steps.read.outputs.host }}","Type":"CNAME","TTL":
300,"ResourceRecords":[{"Value":"$H"}]}}]}
        EOF
        aws route53 change-resource-record-sets --hosted-zone-id "$HZID" --
change-batch file://dns.json
```

> If you're using **GitHub-hosted runners**, add an OIDC step (`aws-actions/configure-aws-credentials`) to assume a role.

---

9) Actions Runner (Self-Hosted) — Make It Hands-Off

1. Install runner on EC2 once.
2. Register with labels: `self-hosted, linux, x64` **(+ optional project label)**.
3. Install as a **service** (`./svc.sh install` → `start`).
4. Ensure the runner shows **Idle** in GitHub → Settings → Actions → Runners.

> If jobs say *"Waiting for a runner..."*: fix labels or re-register the runner with a **fresh token**.

---

10) First Deployment (Happy Path)

1. Commit code + `project.yaml` + templates + `deploy.yml`.

2. Push to `main`.
3. Watch Actions run → build → push → apply.
4. Wait until Ingress gets a hostname.
5. Route53 record is created → open `https://your-host` 🥚

---

11) Troubleshooting (Most Common Oopsies)

- **Ingress hostname never appears**
- Check: `kubectl -n <ns> describe ingress <app>-ingress`

- Ensure subnets/security groups/certificate ARN are correct. ALB controller must be installed.

- **YAML parse error:** `expected alphabetic or numeric character`

- You probably have an unquoted `*` or special character (e.g., password). Use quotes: `"p@ss*word"`.
- Use `---` between multiple YAML docs.

- **Runner says active but jobs don't start**

- Labels mismatch or stale registration. Re-register with correct labels; ensure service is running.

- **Unauthorized** `kubectl`

- Your runner's IAM doesn't map in `aws-auth` or you didn't `aws eks update-kubeconfig`. Fix IAM role mapping.

- **Secrets not loading in pod**

- Check ServiceAccount annotation for IRSA, IAM policy permissions, and pod env/secretKeyRef names.

---

12) Security & Maintenance Tips

- Prefer **IRSA** and **Instance Profiles** over static keys.
- Rotate secrets in **Secrets Manager**; avoid committing real `.env` values.
- Store Terraform state in **S3** with **DynamoDB** lock.
- Pin container base images and regularly rebuild.

---

13) Quick Start (Copy/Paste)

```
# Build & push locally (sanity):
AWS_ACCOUNT=123456789012
REGION=us-east-1
REPO=cicd-microservices
TAG=test
aws ecr get-login-password --region $REGION | \
  docker login --username AWS --password-stdin $AWS_ACCOUNT.dkr.ecr.
$REGION.amazonaws.com

docker build -t $AWS_ACCOUNT.dkr.ecr.$REGION.amazonaws.com/$REPO:$TAG .
docker push    $AWS_ACCOUNT.dkr.ecr.$REGION.amazonaws.com/$REPO:$TAG

# Kube context (from runner or your laptop):
aws eks update-kubeconfig --name dev-eks-2 --region us-east-1

# Render a template (example):
export APP_NAME=myapp NAMESPACE=cicd-myapp ACCOUNT_ID=$AWS_ACCOUNT
REGION=$REGION \
       ECR_REPO=$REPO CONTAINER_PORT=5000 SERVICE_PORT=80 IMAGE_TAG=$TAG \
       HOST=myapp.apps-aws.com CERT_ARN=arn:aws:acm:us-east-1:
$AWS_ACCOUNT:certificate/xxxx \
       INGRESS_CLASS=alb SUBNETS="subnet-aaa,subnet-bbb" SECURITY_GROUPS="sg-
zzz" HEALTHCHECK_PATH=/
mkdir -p k8s
envsubst < cicd/templates/deployment.yaml.tpl > k8s/deployment.yaml
envsubst < cicd/templates/service.yaml.tpl    > k8s/service.yaml
envsubst < cicd/templates/ingress.yaml.tpl    > k8s/ingress.yaml
kubectl -n $NAMESPACE apply -f k8s/
```

---

14) What Makes This Generic?

- All project differences live in `cicd/project.yaml` (and secrets).
- The workflow reads config → renders templates → deploys.
- New project? Copy the template repo, change **one** file, push. Done.

---

15) One-Page Summary (for your team)

- **Code** → GitHub → **Actions** builds Docker → push to **ECR** → **EKS** runs it → **ALB** gets hostname → **Route53** points domain.
- Keep secrets in **Secrets Manager**; use **IRSA**.

• Runner must be **Idle** with correct **labels**.
• YAML docs separated by `---`; quote tricky characters.
• If it’s not working, read section **11) Troubleshooting** first.