Go + Flutter Course

CI/CD & Deployment

Timur Harin

Lecture 07: **CI/CD & Deployment**

From code to production: Automated delivery pipelines

Block 7: CI/CD & Deployment

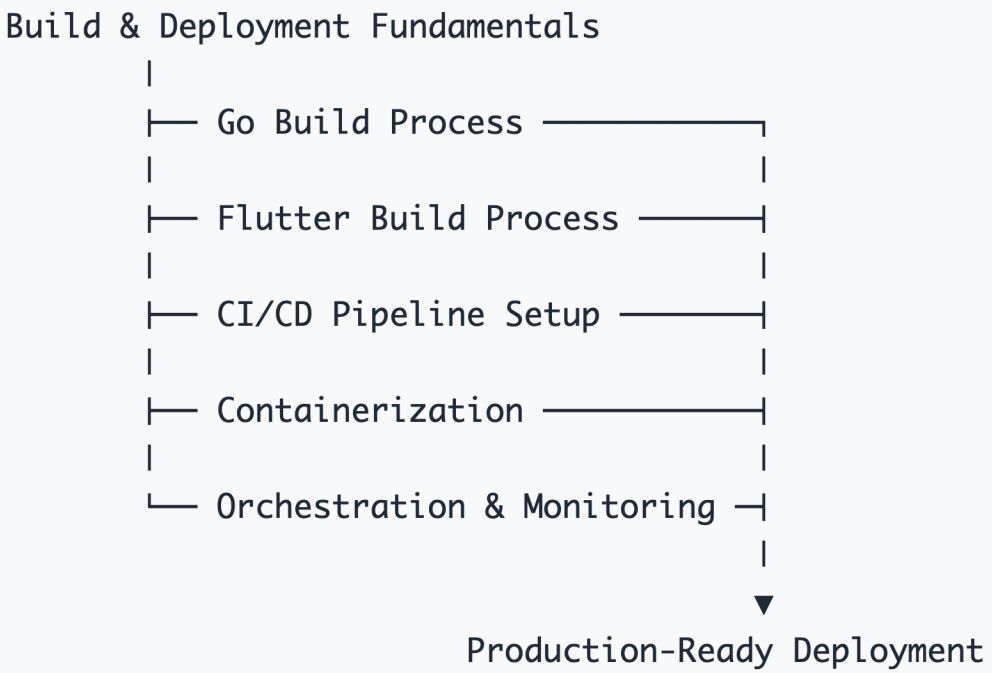
Lecture 07 Overview

- **Build Fundamentals**: Understanding compilation and build processes
- **CI/CD Pipelines**: Automated testing, building, and deployment
- **Containerization**: Docker for Go and Flutter applications
- **Orchestration**: Kubernetes deployment and management
- **Store Deployment**: Publishing to app stores and web platforms

What we'll learn

- **Build processes for Go binaries and Flutter apps**
- **Setting up comprehensive CI/CD pipelines**
- **Container best practices and multi-stage builds**
- **Kubernetes deployment strategies and configurations**
- **App store publishing and web deployment**
- **Monitoring and rollback strategies**

Learning path



- **Foundation**: Understanding build processes and artifacts
- **Automation**: CI/CD pipelines for consistent delivery
- **Containerization**: Packaging applications for deployment
- **Orchestration**: Managing applications at scale

Part I: Build Fundamentals

> ***Build processes*** *transform source code into executable artifacts that can run on target platforms.*

Build pipeline stages

1. **Source Code**: Version-controlled application code
2. **Dependencies**: Package management and dependency resolution
3. **Compilation**: Transform source to binary/bytecode
4. **Testing**: Automated quality assurance
5. **Packaging**: Bundle into deployable artifacts
6. **Distribution**: Deliver to target environments

Key concepts

- **Cross-compilation**: Building for different platforms
- **Build artifacts**: Output files from build process
- **Build tools**: Compilers, bundlers, package managers
- **Environment consistency**: Reproducible builds

Go build fundamentals

Basic Go compilation

```
# Build for current platform
go build -o myapp ./cmd/server

# Build with specific module
go build -o myapp github.com/user/repo/cmd/server

# Build with version information
go build -ldflags "-X main.version=1.0.0" -o myapp

# Optimize for production
go build -ldflags "-s -w" -o myapp
# -s: strip symbol table
# -w: strip debug information
```

Cross-platform compilation

```
# Build for Linux
GOOS=linux GOARCH=amd64 go build -o myapp-linux

# Build for Windows
GOOS=windows GOARCH=amd64 go build -o myapp.exe

# Build for macOS (Intel)
GOOS=darwin GOARCH=amd64 go build -o myapp-darwin

# Build for macOS (Apple Silicon)
GOOS=darwin GOARCH=arm64 go build -o myapp-darwin-arm64

# List all supported platforms
go tool dist list
```

Go build optimization

Build script example

```
#!/bin/bash
# scripts/build.sh

VERSION=$(git describe --tags --always)
BUILD_TIME=$(date -u '+%Y-%m-%d_%H:%M:%S')
COMMIT_HASH=$(git rev-parse HEAD)

LDFLAGS="-s -w \
  -X main.version=${VERSION} \
  -X main.buildTime=${BUILD_TIME} \
  -X main.commitHash=${COMMIT_HASH}"

echo "Building version: ${VERSION}"

# Build for multiple platforms
platforms=("linux/amd64" "darwin/amd64" "windows/amd64")

for platform in "${platforms[@]}"; do
  GOOS=${platform%/*}
  GOARCH=${platform#*/}
  output="dist/myapp-${GOOS}-${GOARCH}"

  if [ "$GOOS" = "windows" ]; then
    output+=".exe"
  fi

  GOOS=$GOOS GOARCH=$GOARCH go build \
    -ldflags "${LDFLAGS}" \
    -o "$output" ./cmd/server
```

Version information injection

```
// cmd/server/main.go
package main

import (
    "fmt"
    "log"
    "runtime"
)

var (
    version    string = "dev"
    buildTime  string = "unknown"
    commitHash string = "unknown"
)

func main() {
    fmt.Printf("App Version: %s\n", version)
    fmt.Printf("Build Time: %s\n", buildTime)
    fmt.Printf("Commit: %s\n", commitHash)
    fmt.Printf("Go Version: %s\n", runtime.Version())
    fmt.Printf("Platform: %s/%s\n", runtime.GOOS, runtime.GOARCH)

    // Start your application
    startServer()
}
```

Flutter build fundamentals

Build modes

```
# Debug mode (development)
flutter run  # Hot reload enabled
flutter build apk --debug

# Profile mode (performance testing)
flutter build apk --profile
flutter run --profile

# Release mode (production)
flutter build apk --release
flutter build appbundle --release  # For Play Store
flutter build ios --release
flutter build web --release
```

Platform-specific builds

```
# Android
flutter build apk --split-per-abi  # Multiple APKs
flutter build appbundle           # App Bundle (recommended)

# iOS
```

Build configuration

```
# pubspec.yaml
flutter:
  uses-material-design: true
  assets:
    - assets/images/
    - assets/configs/

# Build flavors
flutter:
  flavors:
    development:
      android:
        applicationId: "com.example.app.dev"
      ios:
        bundleId: "com.example.app.dev"
    production:
      android:
        applicationId: "com.example.app"
      ios:
        bundleId: "com.example.app"
```

Build artifacts and versioning

Go build artifacts

```
dist/
├── myapp-linux-amd64           # Linux binary
├── myapp-darwin-amd64          # macOS Intel binary
├── myapp-darwin-arm64          # macOS ARM binary
├── myapp-windows-amd64.exe     # Windows binary
├── checksums.sha256            # Integrity verification
└── release-notes.md            # Version information
```

Makefile for consistent builds

```
# Makefile
VERSION := $(shell git describe --tags --always)
LDFLAGS := -s -w -X main.version=$(VERSION)

.PHONY: build
build:
	go build -ldflags "$(LDFLAGS)" -o bin/myapp ./cmd/server

.PHONY: build-all
build-all:
	GOOS=linux GOARCH=amd64 go build -ldflags "$(LDFLAGS)" -o dist/myapp-linux ./cmd/server
	GOOS=darwin GOARCH=amd64 go build -ldflags "$(LDFLAGS)" -o dist/myapp-darwin ./cmd/server
	GOOS=windows GOARCH=amd64 go build -ldflags "$(LDFLAGS)" -o dist/myapp.exe ./cmd/server

.PHONY: test
test:
	go test -v ./...
```

Flutter version management

```
# pubspec.yaml
name: myapp
description: My Flutter application
version: 1.2.3+4  # semantic version + build number

environment:
  sdk: ">=3.0.0 <4.0.0"
  flutter: ">=3.10.0"
```

```
// lib/config/app_config.dart
class AppConfig {
  static const String appName = 'MyApp';
  static const String version = String.fromEnvironment(
    'APP_VERSION',
    defaultValue: '1.0.0',
  );
  static const String buildNumber = String.fromEnvironment(
    'BUILD_NUMBER',
    defaultValue: '1',
  );
  static const String environment = String.fromEnvironment(
```

Part II: CI/CD Pipeline Setup

> ***Continuous Integration/Continuous Deployment*** *automates the software delivery process, ensuring quality and enabling rapid, reliable releases.*

CI/CD benefits

- **Automated testing**: Catch bugs early and consistently
- **Consistent builds**: Same process every time
- **Fast feedback**: Quick detection of issues
- **Reduced manual work**: Less human error
- **Faster releases**: Deploy frequently with confidence

Popular CI/CD platforms

- **GitHub Actions**: Integrated with GitHub repositories
- **GitLab CI/CD**: Built into GitLab platform
- **Jenkins**: Self-hosted, highly customizable
- **CircleCI**: Cloud-based with Docker support
- **Azure DevOps**: Microsoft's complete DevOps platform

GitHub Actions for Go

Basic Go workflow

```yaml
# .github/workflows/go.yml
name: Go CI/CD

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]
  release:
    types: [ created ]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4

    - name: Set up Go
      uses: actions/setup-go@v4
      with:
        go-version: '1.21'

    - name: Cache dependencies
      uses: actions/cache@v3
      with:
        path: ~/go/pkg/mod
        key: ${{ runner.os }}-go-${{ hashFiles('**/go.sum') }}

    - name: Download dependencies
      run: go mod download

    - name: Run tests
      run: go test -v -race -coverprofile=coverage.out ./...
```

Multi-platform build job

```yaml
build:
  needs: test
  runs-on: ubuntu-latest
  strategy:
    matrix:
      goos: [linux, windows, darwin]
      goarch: [amd64, arm64]
      exclude:
        - goos: windows
          goarch: arm64
  steps:
  - uses: actions/checkout@v4

  - name: Set up Go
    uses: actions/setup-go@v4
    with:
      go-version: '1.21'

  - name: Build binary
    env:
      GOOS: ${{ matrix.goos }}
      GOARCH: ${{ matrix.goarch }}
    run: |
      BINARY_NAME=myapp-${{ matrix.goos }}-${{ matrix.goarch }}
      if [ "${{ matrix.goos }}" = "windows" ]; then
        BINARY_NAME+=.exe
      fi
      go build -ldflags "-s -w" -o $BINARY_NAME ./cmd/server

  - name: Upload artifacts
    uses: actions/upload-artifact@v3
```

GitHub Actions for Flutter

Flutter CI workflow

```yaml
# .github/workflows/flutter.yml
name: Flutter CI/CD

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4

    - name: Setup Flutter
      uses: subosito/flutter-action@v2
      with:
        flutter-version: '3.13.0'
        channel: 'stable'

    - name: Install dependencies
      run: flutter pub get

    - name: Run analyzer
      run: flutter analyze

    - name: Run tests
      run: flutter test --coverage

    - name: Upload coverage
```

Android build and deployment

```yaml
build-android:
  needs: test
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v4

  - name: Setup Flutter
    uses: subosito/flutter-action@v2
    with:
      flutter-version: '3.13.0'

  - name: Setup Java
    uses: actions/setup-java@v3
    with:
      distribution: 'zulu'
      java-version: '17'

  - name: Decode signing key
    run: |
      echo "${{ secrets.KEYSTORE_BASE64 }}" | base64 -d > android/app/keystore.jks

  - name: Create key.properties
    run: |
      echo "storePassword=${{ secrets.STORE_PASSWORD }}" > android/key.properties
      echo "keyPassword=${{ secrets.KEY_PASSWORD }}" >> android/key.properties
      echo "keyAlias=${{ secrets.KEY_ALIAS }}" >> android/key.properties
      echo "storeFile=keystore.jks" >> android/key.properties

  - name: Build APK
    run: flutter build apk --release

  - name: Build App Bundle
```

Advanced CI/CD patterns

Matrix testing strategy

```yaml
# .github/workflows/matrix.yml
jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        go-version: ['1.20', '1.21']
        include:
          - os: ubuntu-latest
            go-version: '1.21'
            coverage: true
    steps:
    - uses: actions/checkout@v4

    - name: Set up Go ${{ matrix.go-version }}
      uses: actions/setup-go@v4
      with:
        go-version: ${{ matrix.go-version }}

    - name: Run tests
      run: go test -v ./...

    - name: Run tests with coverage
```

Conditional deployment

```yaml
  deploy:
    needs: [test, build]
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main' && github.event_name == 'push'
    environment:
      name: production
      url: https://myapp.example.com
    steps:
    - name: Deploy to staging
      if: contains(github.ref, 'develop')
      run: echo "Deploying to staging"

    - name: Deploy to production
      if: github.ref == 'refs/heads/main'
      run: echo "Deploying to production"

    - name: Notify deployment
      uses: 8398a7/action-slack@v3
      with:
        status: ${{ job.status }}
        channel: '#deployments'
        webhook_url: ${{ secrets.SLACK_WEBHOOK }}
```

Part III: Containerization

> ***Docker containers*** *provide consistent, portable deployment environments by packaging applications with their dependencies.*

Container benefits

- **Consistency**: Same environment everywhere
- **Isolation**: Applications don't interfere with each other
- **Portability**: Run anywhere Docker is supported
- **Scalability**: Easy to scale up/down
- **Resource efficiency**: Lower overhead than VMs

Docker concepts

- **Image**: Read-only template for creating containers
- **Container**: Running instance of an image
- **Dockerfile**: Instructions for building images
- **Registry**: Repository for storing and sharing images

Docker for Go applications

Basic Dockerfile

```dockerfile
# Dockerfile
FROM golang:1.21-alpine AS builder

WORKDIR /app
COPY go.mod go.sum ./
RUN go mod download

COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build \
    -ldflags "-s -w" \
    -o main ./cmd/server

FROM alpine:latest
RUN apk --no-cache add ca-certificates tzdata
WORKDIR /root/

COPY --from=builder /app/main .

EXPOSE 8080
CMD ["./main"]
```

Multi-stage optimized build

```dockerfile
# Dockerfile.optimized
FROM golang:1.21-alpine AS builder

# Install git for private modules
RUN apk add --no-cache git

WORKDIR /app

# Copy dependency files first for better caching
COPY go.mod go.sum ./
RUN go mod download

# Copy source code
COPY . .

# Build the binary
RUN CGO_ENABLED=0 GOOS=linux GOARCH=amd64 go build \
    -ldflags="-w -s -X main.version=${VERSION}" \
    -a -installsuffix cgo \
    -o main ./cmd/server

# Final stage
FROM scratch

# Add CA certificates for HTTPS
COPY --from=builder /etc/ssl/certs/ca-certificates.crt /etc/ssl/certs/

# Add timezone data
COPY --from=builder /usr/share/zoneinfo /usr/share/zoneinfo

# Add binary
COPY --from=builder /app/main /main
```

Docker for Flutter Web

Flutter web Dockerfile

```dockerfile
# Dockerfile.flutter-web
FROM cirrusci/flutter:stable AS builder

WORKDIR /app
COPY pubspec.* ./
RUN flutter pub get

COPY . .
RUN flutter build web --release

FROM nginx:alpine
COPY --from=builder /app/build/web /usr/share/nginx/html

# Custom nginx configuration
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Nginx configuration

```nginx
# nginx.conf
events {
    worker_connections 1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    gzip on;
    gzip_vary on;
    gzip_min_length 1024;
    gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript;

    server {
        listen 80;
        server_name localhost;
        root /usr/share/nginx/html;
        index index.html;

        # Handle Flutter routing
        location / {
            try_files $uri $uri/ /index.html;
        }

        # Cache static assets
        location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2)$ {
            expires 1y;
            add_header Cache-Control "public, immutable";
        }
    }
}
```

Docker Compose for development

Development setup

```yaml
# docker-compose.yml
version: '3.8'

services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile.dev
    ports:
      - "8080:8080"
    environment:
      - DB_HOST=postgres
      - DB_NAME=myapp_dev
      - DB_USER=postgres
      - DB_PASSWORD=password
      - REDIS_URL=redis:6379
    volumes:
      - ./backend:/app
      - go_modules:/go/pkg/mod
    depends_on:
      - postgres
      - redis
    restart: unless-stopped

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile.dev
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
      - node_modules:/app/node_modules
```

Database and services

```yaml
  postgres:
    image: postgres:15-alpine
    environment:
      - POSTGRES_DB=myapp_dev
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=password
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./backend/migrations:/docker-entrypoint-initdb.d
    ports:
      - "5432:5432"
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - backend
      - frontend
    restart: unless-stopped

volumes:
  postgres_data:
```

Container best practices

Security best practices

```
# Use specific version tags
FROM golang:1.21.4-alpine3.18

# Create non-root user
RUN addgroup -g 1001 -S appgroup && \
    adduser -u 1001 -S appuser -G appgroup

# Install only necessary packages
RUN apk add --no-cache ca-certificates tzdata

# Use non-root user
USER appuser

# Set read-only filesystem
COPY --chown=appuser:appgroup --from=builder /app/main /main

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD wget --no-verbose --tries=1 --spider http://localhost:8080/health || exit 1
```

Performance optimization

```
# Minimize layers
RUN apk add --no-cache \
    ca-certificates \
    tzdata \
    && rm -rf /var/cache/apk/*

# Use .dockerignore
# .dockerignore
.git
.gitignore
README.md
Dockerfile*
docker-compose*
.dockerignore
.env*
coverage.txt
*.md
tests/
.github/
```

```
# Build with build context optimization
```

Part IV: Kubernetes Deployment

> ***Kubernetes*** *orchestrates containerized applications at scale, providing automated deployment, scaling, and management.*

Kubernetes core concepts

- **Pod**: Smallest deployable unit (one or more containers)
- **Deployment**: Manages replica sets and rolling updates
- **Service**: Network endpoint for accessing pods
- **ConfigMap**: Configuration data storage
- **Secret**: Sensitive data storage
- **Ingress**: External access management

Benefits

- **Self-healing**: Automatic pod replacement and health monitoring
- **Scaling**: Horizontal and vertical pod autoscaling
- **Rolling updates**: Zero-downtime deployments
- **Service discovery**: Built-in load balancing and networking

Basic Kubernetes manifests

Deployment configuration

```yaml
# k8s/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-backend
  labels:
    app: myapp-backend
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp-backend
  template:
    metadata:
      labels:
        app: myapp-backend
    spec:
      containers:
      - name: backend
        image: myapp/backend:v1.0.0
        ports:
        - containerPort: 8080
        env:
        - name: DB_HOST
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: db-host
        - name: DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: app-secrets
              key: db-password
        resources:
          limits:
            cpu: 500m
```

Service configuration

```yaml
# k8s/service.yaml
apiVersion: v1
kind: Service
metadata:
  name: myapp-backend-service
spec:
  selector:
    app: myapp-backend
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
  type: ClusterIP

---
# k8s/configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
data:
  db-host: "postgres-service"
  db-name: "myapp"
  redis-url: "redis-service:6379"

---
# k8s/secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
type: Opaque
```

Ingress and external access

Ingress configuration

```yaml
# k8s/ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: myapp-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/rate-limit: "100"
spec:
  tls:
  - hosts:
    - api.myapp.com
    - myapp.com
    secretName: myapp-tls
  rules:
  - host: api.myapp.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: myapp-backend-service
            port:
              number: 80
  - host: myapp.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
```

Horizontal Pod Autoscaler

```yaml
# k8s/hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: myapp-backend-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: myapp-backend
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleUp:
      stabilizationWindowSeconds: 60
      policies:
      - type: Percent
        value: 100
        periodSeconds: 15
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
```

Health checks and monitoring

Pod health checks

```yaml
# Enhanced deployment with health checks
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-backend
spec:
  template:
    spec:
      containers:
      - name: backend
        image: myapp/backend:v1.0.0
        ports:
        - containerPort: 8080
        livenessProbe:
          httpGet:
            path: /health/live
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3
        readinessProbe:
          httpGet:
            path: /health/ready
            port: 8080
          initialDelaySeconds: 5
          periodSeconds: 5
          timeoutSeconds: 3
          failureThreshold: 2
        startupProbe:
          httpGet:
            path: /health/startup
```

Go health check endpoints

```go
// internal/handlers/health.go
func HealthHandler(db *sql.DB, redis *redis.Client) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        health := map[string]interface{}{
            "status":    "healthy",
            "timestamp": time.Now().UTC(),
            "version":   version,
        }

        // Check database
        if err := db.Ping(); err != nil {
            health["status"] = "unhealthy"
            health["database"] = "down"
            w.WriteHeader(http.StatusServiceUnavailable)
        } else {
            health["database"] = "up"
        }

        // Check Redis
        if err := redis.Ping().Err(); err != nil {
            health["status"] = "unhealthy"
            health["redis"] = "down"
            w.WriteHeader(http.StatusServiceUnavailable)
        } else {
            health["redis"] = "up"
        }
```

Deployment strategies

Rolling update strategy

```yaml
# k8s/deployment-rolling.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-backend
spec:
  replicas: 5
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      annotations:
        # Force pods to restart on config changes
        configmap.reloader.stakater.com/reload: "app-config"
    spec:
      containers:
      - name: backend
        image: myapp/backend:v1.1.0
        # Graceful shutdown
        lifecycle:
          preStop:
            exec:
```

Blue-green deployment

```yaml
# Blue deployment (current)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-backend-blue
  labels:
    version: blue
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp-backend
      version: blue

---
# Green deployment (new)
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-backend-green
  labels:
    version: green
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp-backend
      version: green

---
# Switch traffic by updating service selector
apiVersion: v1
kind: Service
metadata:
  name: myapp-backend-service
```

Part V: App Store Deployment

> ***App store deployment*** *involves specific requirements, code signing, and review processes for different platforms.*

Platform requirements

Platform	Format	Requirements
Google Play	AAB/APK	Google Play Console account, signing key
Apple App Store	IPA	Apple Developer account, certificates
Web Hosting	Static files	Domain, SSL certificate
Microsoft Store	MSIX	Microsoft Partner Center account

Key considerations

- **Code signing**: Cryptographic verification of app authenticity
- **App review**: Platform-specific guidelines and approval process
- **Release management**: Staged rollouts and version control
- **Metadata**: App descriptions, screenshots, privacy policy

Android deployment setup

Generate signing key

```
# Create keystore for app signing
keytool -genkey -v -keystore ~/upload-keystore.jks \
  -keyalg RSA -keysize 2048 -validity 10000 \
  -alias upload

# Create key.properties file
echo "storePassword=myStorePassword" > android/key.properties
echo "keyPassword=myKeyPassword" >> android/key.properties
echo "keyAlias=upload" >> android/key.properties
echo "storeFile=/Users/username/upload-keystore.jks" >> android/key.properties
```

Configure build.gradle

```
// android/app/build.gradle
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
    keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
}

android {
    signingConfigs {
        release {
            keyAlias keystoreProperties['keyAlias']
            keyPassword keystoreProperties['keyPassword']
            storeFile keystoreProperties['storeFile'] ? file(keystoreProperties['storeFile']) : null
            storePassword keystoreProperties['storePassword']
        }
    }
    buildTypes {
        release {
```

CI/CD for Play Store

```
# .github/workflows/android-deploy.yml
name: Deploy to Play Store

on:
  push:
    tags:
      - 'v*'

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4

    - name: Setup Flutter
      uses: subosito/flutter-action@v2
      with:
        flutter-version: '3.13.0'

    - name: Create key.properties
      run: |
        echo "storePassword=${{ secrets.STORE_PASSWORD }}" > android/key.properties
        echo "keyPassword=${{ secrets.KEY_PASSWORD }}" >> android/key.properties
        echo "keyAlias=${{ secrets.KEY_ALIAS }}" >> android/key.properties
        echo "storeFile=keystore.jks" >> android/key.properties

    - name: Decode keystore
      run: echo "${{ secrets.KEYSTORE_BASE64 }}" | base64 -d > android/app/keystore.jks

    - name: Build App Bundle
      run: flutter build appbundle --release

    - name: Deploy to Play Store
      uses: r0adkll/upload-google-play@v1
      with:
        serviceAccountJsonPlainText: ${{ secrets.SERVICE_ACCOUNT_JSON }}
```

iOS deployment setup

Certificate management

```
# Install certificates (manual)
# 1. Download certificates from Apple Developer Portal
# 2. Install in Keychain Access
# 3. Create provisioning profiles

# Or use Fastlane for automation
gem install fastlane

# Initialize Fastlane
cd ios && fastlane init

# Create Appfile
echo "app_identifier 'com.example.myapp'" > fastlane/Appfile
echo "apple_id 'your-apple-id@example.com'" >> fastlane/Appfile
echo "team_id 'YOUR_TEAM_ID'" >> fastlane/Appfile
```

Fastlane configuration

```
# ios/fastlane/Fastfile
default_platform(:ios)

platform :ios do
```

GitHub Actions for iOS

```
# .github/workflows/ios-deploy.yml
name: Deploy iOS to App Store

on:
  push:
    tags:
      - 'v*'

jobs:
  deploy:
    runs-on: macos-latest
    steps:
    - uses: actions/checkout@v4

    - name: Setup Flutter
      uses: subosito/flutter-action@v2
      with:
        flutter-version: '3.13.0'

    - name: Install Apple certificates
      uses: apple-actions/import-codesign-certs@v2
      with:
        p12-file-base64: ${{ secrets.CERTIFICATES_P12 }}
        p12-password: ${{ secrets.CERTIFICATES_PASSWORD }}

    - name: Install provisioning profile
      uses: apple-actions/download-provisioning-profiles@v1
      with:
        bundle-id: com.example.myapp
        issuer-id: ${{ secrets.APPSTORE_ISSUER_ID }}
        api-key-id: ${{ secrets.APPSTORE_KEY_ID }}
        api-private-key: ${{ secrets.APPSTORE_PRIVATE_KEY }}

    - name: Build and deploy
      run: |
```

Web deployment strategies

Static hosting (Netlify/Vercel)

```
# netlify.toml
[build]
  publish = "build/web"
  command = "flutter build web --release"

[build.environment]
  FLUTTER_VERSION = "3.13.0"

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200

[headers]
  for = "/*"
  [headers.values]
    X-Frame-Options = "DENY"
    X-XSS-Protection = "1; mode=block"
    X-Content-Type-Options = "nosniff"
```

Docker deployment

```
# Multi-stage Flutter web build
FROM cirrusci/flutter:stable AS builder

WORKDIR /app
COPY pubspec.* ./
RUN flutter pub get

COPY . .
RUN flutter build web --release

FROM nginx:alpine
COPY --from=builder /app/build/web /usr/share/nginx/html
COPY nginx.conf /etc/nginx/nginx.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

CDN deployment

Kubernetes deployment

```
apiVersion: apps/v1
```

Monitoring and observability

Application metrics

```
// internal/monitoring/metrics.go
import (
    "github.com/prometheus/client_golang/prometheus"
    "github.com/prometheus/client_golang/prometheus/promhttp"
)

var (
    httpRequestsTotal = prometheus.NewCounterVec(
        prometheus.CounterOpts{
            Name: "http_requests_total",
            Help: "Total number of HTTP requests",
        },
        []string{"method", "endpoint", "status"},
    )

    httpRequestDuration = prometheus.NewHistogramVec(
        prometheus.HistogramOpts{
            Name: "http_request_duration_seconds",
            Help: "HTTP request duration in seconds",
        },
        []string{"method", "endpoint"},
    )
)

func init() {
    prometheus.MustRegister(httpRequestsTotal)
    prometheus.MustRegister(httpRequestDuration)
}
```

Logging setup

```
// internal/logging/logger.go
import (
    "go.uber.org/zap"
    "go.uber.org/zap/zapcore"
)

func NewLogger(level string, environment string) (*zap.Logger, error) {
    var config zap.Config

    if environment == "production" {
        config = zap.NewProductionConfig()
        config.DisableStacktrace = true
    } else {
        config = zap.NewDevelopmentConfig()
    }

    config.Level = zap.NewAtomicLevelAt(parseLevel(level))
    config.EncoderConfig.TimeKey = "timestamp"
    config.EncoderConfig.EncodeTime = zapcore.ISO8601TimeEncoder

    return config.Build()
}

func parseLevel(level string) zapcore.Level {
    switch level {
    case "debug": return zapcore.DebugLevel
    case "info": return zapcore.InfoLevel
    case "warn": return zapcore.WarnLevel
    case "error": return zapcore.ErrorLevel
```

What we've learned

Build Fundamentals

- **Go compilation**: Cross-platform builds and optimization techniques
- **Flutter builds**: Platform-specific artifacts and build modes
- **Version management**: Embedding build information and semantic versioning
- **Build automation**: Makefiles and scripts for consistent builds

CI/CD Pipelines

- **GitHub Actions**: Automated testing and deployment workflows
- **Matrix strategies**: Testing across multiple platforms and versions
- **Conditional deployment**: Environment-specific release strategies
- **Security**: Secrets management and secure artifact handling

What we've learned (continued)

Containerization

- **Docker fundamentals**: Images, containers, and multi-stage builds
- **Go containerization**: Optimized Docker images with scratch base
- **Flutter web**: Nginx-based container deployment
- **Best practices**: Security, optimization, and health checks

Kubernetes Orchestration

- **Core concepts**: Pods, Deployments, Services, and Ingress
- **Scaling**: Horizontal Pod Autoscaler and resource management
- **Health monitoring**: Liveness, readiness, and startup probes
- **Deployment strategies**: Rolling updates and blue-green deployments

App Store Deployment

- **Android**: Google Play Store publishing with AAB format
- **iOS**: App Store Connect deployment with code signing

Thank You!

What's Next:

- Lab 07: Implement complete CI/CD pipeline for your project

Resources:

- Docker Best Practices: https://docs.docker.com/develop/dev-best-practices/
- Kubernetes Documentation: https://kubernetes.io/docs/
- GitHub Actions: https://docs.github.com/en/actions
- Flutter Deployment: https://docs.flutter.dev/deployment
- Course Repository: https://github.com/timur-harin/sum25-go-flutter-course

Contact:

- Email: timur.harin@mail.com
- Telegram: @timur_harin

Next Lecture: Final Project & Wrap-up

Questions?