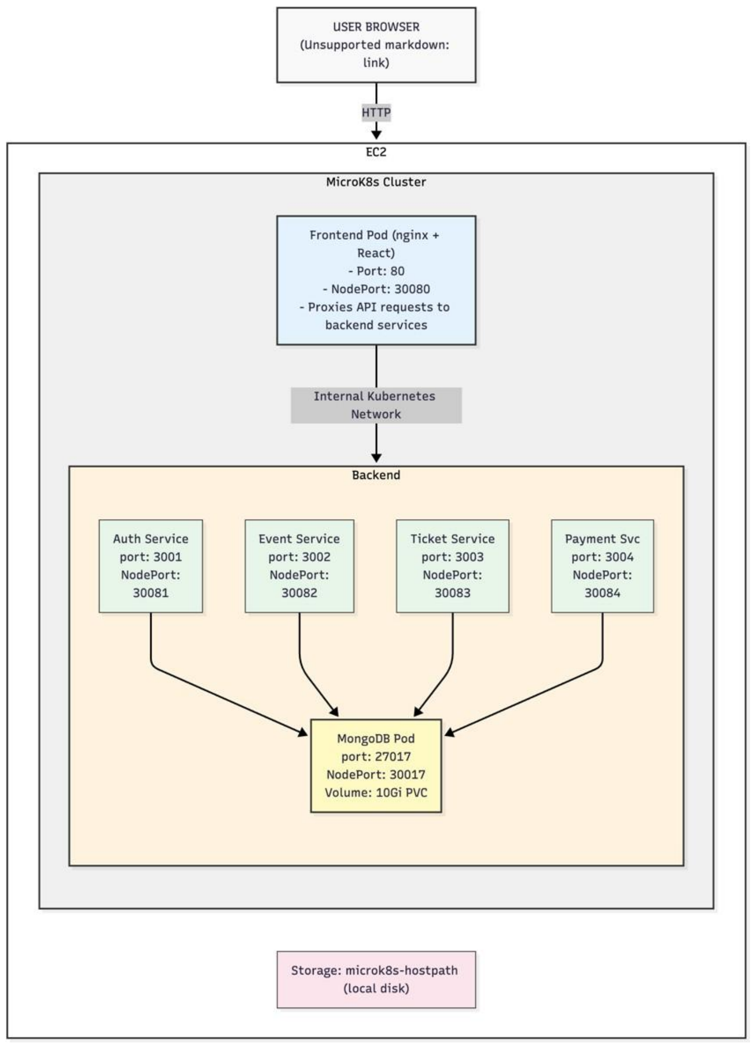
Ticketer - Event Management & Ticket Booking Platform

Architecture



USER BROWSER (Unsupported markdown: link)
HTTP
EC2
MicroK8s Cluster
Frontend Pod (nginx + React) - Port: 80 - NodePort: 30080 - Proxies API requests to backend services
Internal Kubernetes Network
Backend
Auth Service port: 3001 NodePort: 30081
Event Service port: 3002 NodePort: 30082
Ticket Service port: 3003 NodePort: 30083
Payment Svc port: 3004 NodePort: 30084
MongoDB Pod port: 27017 NodePort: 30017 Volume: 10Gi PVC
Storage: microk8s-hostpath (local disk)

Technology Stack

Frontend

- Framework: React 18
- Build Tool: Vite
- Styling: Tailwind CSS

Backend Services

- Runtime: Node.js 18 (Alpine Linux)
- Framework: Express.js
- Session: express-session + connect-mongo
- Authentication: Session-based (cookies)

Database

- Database: MongoDB 7.0
- ODM: Mongoose
- Storage: Persistent Volume (10Gi)

DevOps

- Containerization: Docker (multi-stage builds)
- Registry: GitHub Container Registry (GHCR)
- Orchestration: MicroK8s 1.29
- Cloud Provider: AWS EC2
- Infrastructure as Code: Terraform
- CI/CD: GitHub Actions

Networking

- Reverse Proxy: Nginx (in frontend container)
- Service Discovery: Kubernetes DNS
- External Access: NodePort (30080-30084)
- Internal Communication: ClusterIP

Microservices

1. Auth Service (Port 3001)

Purpose: User authentication and session management

Endpoints:

- `POST /api/auth/register` - Register new user
- `POST /api/auth/login` - Login user
- `POST /api/auth/logout` - Logout user
- `GET /api/auth/me` - Get current user
- `GET /api/auth/verify` - Verify session (internal)

Database Collections:

- `users` - User accounts
- `sessions` - Active sessions

2. Event Service (Port 3002)

Purpose: Event creation and management

Endpoints:

- `GET /api/events` · List all events (with filters)
- `GET /api/events/:id` · Get event details
- `POST /api/events` · Create event (authenticated)
- `PUT /api/events/:id` · Update event (authenticated)
- `DELETE /api/events/:id` - Delete event (authenticated)
- `GET /api/events/my/events` - Get user's events

Database Collections:

- `events` - Event listings

3. Ticket Service (Port 3003)

Purpose: Ticket booking and order management

Endpoints:

- `POST /api/tickets/book` - Book tickets (authenticated)
- `GET /api/tickets/orders` - Get user orders
- `GET /api/tickets/orders/:id` - Get order details
- `PUT /api/tickets/orders/:id` - Update order status

Database Collections:

- `orders` - Ticket bookings
- `tickets` - Individual tickets

4. Payment Service (Port 3004)

Purpose: Payment processing

Endpoints:

- `POST /api/payments/process` - Process payment (authenticated)
- `GET /api/payments/history` - Get payment history
- `GET /api/payments/:id` - Get payment details

Database Collections:

`payments` - Payment transactions

5. Frontend (Port 80 → NodePort 30080)

Purpose: User interface and API gateway

Pages:

- `/` - Home/Event listing
- `/login` - User login
- `/events/:id` - Event details
- `/events/create` - Create event (authenticated)
- `/events/my` - My events (authenticated)
- `/orders` - My orders (authenticated)
- `/profile` - User profile (authenticated)

Docker Setup

Container Architecture

Each service uses a multi-stage or optimized Dockerfile:

Backend Services Dockerfile Pattern

```
FROM node:18-alpine

WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm install --production

# Copy source code
COPY . .

# Create non-root user
RUN addgroup -g 1001 -S nodejs && \
    adduser -S nodejs -u 1001 && \
    chown -R nodejs:nodejs /app

USER nodejs

# Expose port
EXPOSE 3001

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
    CMD node healthcheck.js

# Start server
CMD ["node", "src/server.js"]
```

```
# Build stage
FROM node:18-alpine AS builder

WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine

# Copy nginx config
COPY nginx.conf /etc/nginx/conf.d/default.conf

# Copy built assets
COPY --from=builder /app/dist /usr/share/nginx/html

EXPOSE 80

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

Key Points:

- Multi-stage build (smaller image)
- Build artifacts only (no source)
- Nginx for serving + proxying
- Final image: ~50MB

Docker Images

All images are stored in GitHub Container Registry (GHCR):

```
ghcr.io/Anni1808/ticketer-auth:latest
ghcr.io/Anni1808/ticketer-event:latest
ghcr.io/Anni1808/ticketer-ticket:latest
ghcr.io/Anni1808/ticketer-payment:latest
ghcr.io/Anni1808/ticketer-frontend:latest
```

Kubernetes Deployment

Kubernetes Objects

The application uses the following Kubernetes resources:

1. Namespace

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: ticketer
```

All resources are deployed in the `ticketer` namespace for isolation.

2. ConfigMap

File: `k8s/configmap.yaml`

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name:    app-config
  namespace: ticketer
data:
  MONGODB_URI_BASE: "mongodb://admin:PASSWORD@mongodb:27017"

  # Internal service URLs (DNS-based)
  AUTH_SERVICE_URL: "http://auth-service:3001"
  EVENT_SERVICE_URL: "http://event-service:3002"
  TICKET_SERVICE_URL: "http://ticket-service:3003"
  PAYMENT_SERVICE_URL: "http://payment-service:3004"

  NODE_ENV: "production"

  # External frontend URL (replaced at deploy time)
  FRONTEND_URL: "http://EXTERNAL_IP:30080"
```

Key Points:

- Internal URLs use Kubernetes DNS
- `EXTERNAL_IP` is a placeholder (replaced by deploy script)
- All services share this configuration

3. Secret

File: `k8s/secrets.yaml`

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
  namespace: ticketer
type: Opaque
stringData:
  MONGODB_PASSWORD: "your-secure-password"
  SESSION_SECRET: "your-session-secret"
  JWT_SECRET: "your-jwt-secret"
```

Security: Encoded in base64, should be managed securely.

4. PersistentVolumeClaim (MongoDB)

File: `k8s/mongodb.yaml`

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mongodb-pvc
  namespace: ticketer
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage:  10Gi
  storageClassName: microk8s-hostpath
```

Key Points:

- 10Gi storage for database
- `microk8s-hostpath` storage class (local disk)
- Data persists across pod restarts

5. Deployments

Each service has a deployment. Example (Auth Service):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-service
  namespace: ticketer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: auth-service
  template:
     metadata:
       labels:
         app: auth-service
    spec:
      containers:
        - name: auth
          image: ghcr.io/Anni1808/ticketer-auth:latest
          ports:
            - containerPort: 3001
          env:
            - name: PORT
              value: "3001"
          envFrom:
            - configMapRef:
                name: app-config
            - secretRef:
                name: app-secrets
          resources:
            requests:
              memory: "256Mi"
              cpu: "250m"
            limits:
              memory: "512Mi"
              cpu: "500m"
```

Key Points:

- Single replica (can scale horizontally)
- Pulls from GHCR
- Environment from ConfigMap + Secrets
- Resource limits set

6. Services

Each deployment has a corresponding service:

```
apiVersion: v1
kind: Service
metadata:
  name: auth-service
  namespace: ticketer
spec:
  type: NodePort
  selector:
    app: auth-service
  ports:
    - protocol: TCP
      port: 3001 # Internal ClusterIP port
      targetPort: 3001 # Container port
      nodePort: 30081 # External access port
```

Service Types:

- Type: NodePort (exposes on EC2 host)
- ClusterIP: Automatic (internal DNS: `auth-service:3001`)
- NodePort: Manual (external access: `<IP>:30081`)

Port Mapping:

Service	ClusterIP	NodePort
Frontend	frontend:80	30080
Auth	auth-service:3001	30081
Event	event-service:3002	30082
Ticket	ticket-service:3003	30083
Payment	payment-service:3004	30084
MongoDB	mongodb:27017	30017

Kubernetes DNS Resolution

Internal service discovery works via DNS:

```
auth-service                        → ClusterIP (e.g., 10.152.183.45)
auth-service.ticketer               → Same, with namespace
auth-service.ticketer.svc           → Same, with service type
auth-service.ticketer.svc.cluster.local → Full FQDN
```

Services call each other using short names:

- `http://auth-service:3001`
- `http://event-service:3002`
- `mongodb://mongodb:27017`

AWS Infrastructure

EC2 Instance Specifications

Instance Type: t3.small

- vCPUs: 2
- Memory: 2 GB
- Network: Up to 5 Gigabit
- Storage: 20 GB EBS (gp3)

Operating System: Ubuntu 22.04 LTS

Why t3.small?

- Balanced CPU/memory for microservices
- Sufficient for development/small production
- Free tier eligible (750 hours/month, first year)
- Cost: ~$15-17/month after free tier

Network Configuration

Security Group Rules:

Type	Protocol	Port Range	Source	Purpose
SSH	TCP	22	Your IP	Remote access
HTTP	TCP	30080	0.0.0.0/0	Frontend
Custom	TCP	30081-30084	0.0.0.0/0	Backend APIs (optional)
Custom	TCP	30017	Your IP	MongoDB (restricted)

Elastic IP:

- Static IP address
- Persists across instance stop/start
- Changes on destroy/recreate (handled automatically)

Terraform Infrastructure

Directory: `terraform/`

Main Resources:

- VPC: Default VPC
- Security Group: Allow ports 22, 30080-30084, 30017
- EC2 Instance: t3.small with Ubuntu 22.04
- Elastic IP: Associated with instance
- Key Pair: SSH key for access

Deploy Infrastructure:

```
cd terraform
terraform init
terraform plan
terraform apply
```

Outputs:

```
terraform output instance_public_ip  # Get EC2 IP
terraform output instance_id          # Get instance ID
```

CI/CD Pipeline

GitHub Actions Workflow

File: `.github/workflows/ci.yml`

Triggers:

- Push to `main` branch
- Pull requests to `main`
- Manual dispatch

Jobs:

1. Build and Push Images

```
jobs:
  build-and-push:
    runs-on: ubuntu-latest
    timeout-minutes: 30
    permissions:
      contents: read
      packages: write

    strategy:
      fail-fast: false
      matrix:
        include:
          - service: auth
            context: ./auth
            dockerfile: ./auth/Dockerfile
          # ... other services

    steps:
      - Checkout code
      - Set up Docker Buildx
      - Login to GHCR
      - Build and push image
```

What it does:

1. Checks out code
2. Logs into GitHub Container Registry
3. Builds Docker image for `linux/amd64`
4. Pushes to `ghcr.io/Anni1808/ticketer-SERVICE:latest`
5. Also tags with commit SHA