Unleash Feature Management Documentation

Table of Contents

1. Introduction
2. What is Unleash?
3. Use Cases
4. Key Features
5. Local Setup for React Applications
6. Advanced Configuration
7. Best Practices
8. References

Introduction

Unleash is an open-source feature management platform that empowers teams to control feature releases and experiment with new functionality without deploying code. This documentation guides you through understanding Unleash and implementing it in your local React-based applications[1].

What is Unleash?

Unleash is a robust, open-source feature flag management system designed for modern software development teams. It provides a centralized platform to manage feature toggles (also called feature switches or feature flags) across multiple applications, environments, and services[1].

Core Concept

A feature flag is a software development technique that allows you to turn features on or off in real-time without deploying code. Think of it as a universal remote control that manages which "lights" (features) are turned on or off for different users[3].

How It Works

Unleash operates through a client-server architecture:

- **Unleash Server**: Centralized control hub where you define and manage all feature flags, strategies, and configurations
- **Unleash Client SDKs**: Integration libraries that connect your applications (frontend and backend) to the Unleash server
- **Unleash Proxy**: For client-side (browser) applications, an intermediary that securely communicates with the Unleash server without exposing sensitive tokens[1]

Use Cases

1. Progressive Feature Rollouts

Deploy new features to a small percentage of users before releasing to everyone. Monitor behavior, gather feedback, and gradually increase user exposure[1].

Day 1: Enable feature for 5% of users
Day 3: Enable feature for 25% of users
Day 7: Enable feature for 100% of users

2. A/B Testing & Experimentation

Test different feature implementations with different user segments to determine which version performs better without requiring separate deployments[1].

3. Kill Switches

Instantly disable problematic features in production without redeploying code. Provides immediate rollback capability if issues arise[1].

4. User Segmentation

Target features to specific user groups based on:

- Geographic region
- User subscription plan
- Email domain
- Custom properties
- User ID targeting[1]

5. Canary Releases

Release changes to a small subset of users (the "canary") to detect problems early before they impact the entire user base[1].

6. Trunk-Based Development

Enable developers to merge code to the main branch frequently without affecting production users by hiding incomplete features behind flags[1].

7. Blue-Green Deployments

Maintain two live environments and seamlessly switch traffic between them using feature flags to control which environment users access[1].

8. Development & Testing

Enable features for development and staging environments while keeping them disabled in production until ready[1].

Key Features

Feature Flags

Simple on/off toggles for features across your application.

Activation Strategies

Define complex rules for when features should be active:

- **Standard On/Off**: Always enabled or disabled
- **User ID Targeting**: Enable for specific users
- **Gradual Rollout**: Enable for a percentage of users (e.g., 25%)
- **Scheduled Rollout**: Enable features at specific times
- **Custom Strategies**: Define your own targeting logic[1]

Multi-Environment Support

Manage separate feature configurations for development, staging, and production environments with isolated API tokens[1].

Real-Time Evaluation

Feature flags are evaluated instantly on the client or server side, providing immediate control[1].

Instant Rollback

Disable or roll back features with a single click in the dashboard without redeploying code[1].

Audit Trail

Track all changes to feature flags with timestamps and user information for compliance and debugging[1].

Local Setup for React Applications

This section provides a step-by-step guide to set up Unleash locally for React development.

Prerequisites

Ensure you have the following installed:

- **Docker**: Download from docker.com
- **Docker Compose**: Usually included with Docker Desktop
- **Node.js**: Version 16 or higher
- **npm** or **yarn**: Package manager for your React project
- **Git**: For cloning repositories

Step 1: Start Unleash Server Locally

The easiest way to run Unleash locally is using Docker and Docker Compose.

Option A: Using Docker Compose (Recommended)

Clone the Unleash repository and start it with Docker Compose:

Clone the Unleash repository

git clone https://github.com/Unleash/unleash.git

Navigate to the directory

cd unleash

Start Unleash using Docker Compose

docker compose up -d

This command starts:

- **Unleash Server**: Running on http://localhost:4242
- **PostgreSQL Database**: Running on localhost:5432

Option B: Manual Docker Setup

If you prefer to set up containers individually:

Create PostgreSQL container

docker run --name unleash-postgres
-e POSTGRES_USER=unleash
-e POSTGRES_PASSWORD=unleash
-e POSTGRES_DB=unleash
-p 5432:5432
-d postgres

Create Unleash server container

docker run --name unleash-server
-e DATABASE_URL=postgres://unleash:unleash@unleash-postgres:5432/unleash
-p 4242:4242
--link unleash-postgres:unleash-postgres
-d unleashorg/unleash-server

Step 2: Access the Unleash Dashboard

Once the containers are running, open your browser and navigate to:

http://localhost:4242

You'll see the Unleash login screen with default credentials:

- **Username**: `admin`
- **Password**: `unleash4all`

Step 3: Create a Feature Flag in Unleash Dashboard

1. **Log in** with admin credentials
2. **Create a new project** (or use the default one)
3. **Create a new feature flag**:
   - Click "Create feature toggle"
   - Name it: `newFeature`
   - Add a description (optional)
   - Click "Create"
4. **Enable the feature flag**:
   - In the feature toggle details page, click the toggle to enable it
   - Add an activation strategy (e.g., "Percentage Rollout" at 100%)

Step 4: Create a Client Token

1. In the Unleash dashboard, go to **Admin** → **API tokens**
2. Click **"Create API token"**
3. Configure:
   - **Environment**: Select your environment
   - **Token name**: `react-local-token`
   - **Type**: Client
4. Click **Create** and copy the token value

Step 5: Set Up Your React Application

Install Dependencies

Using npm

npm install @unleash/proxy-client-react unleash-proxy-client

Using yarn

yarn add @unleash/proxy-client-react unleash-proxy-client

Configure Unleash in Your React App

In your main application file (typically index.jsx or main.jsx):

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import { FlagProvider } from '@unleash/proxy-client-react';
import App from './App';

const config = {
url: 'http://localhost:4242/api/frontend',
clientKey: 'YOUR_CLIENT_TOKEN_HERE',
appName: 'my-react-app',
};

ReactDOM.createRoot(document.getElementById('root')).render(
<React.StrictMode>


</React.StrictMode>,
);
```

Important: Replace YOUR_CLIENT_TOKEN_HERE with the token you created in Step 4.

Step 6: Use Feature Flags in Components

Use the useFlag hook to check if a feature is enabled:

```
import { useFlag } from '@unleash/proxy-client-react';

function NotificationFeature() {
const isEnabled = useFlag('newFeature');

return (

{isEnabled ? (
```

▯ New Feature Available!

```
You have access to our brand new notification system.


) : (
This feature is coming soon...


)}

);
}

export default NotificationFeature;
```

Step 7: Verify the Setup

1. Start your React development server:

```
npm run dev
```

or

```
yarn dev
```

2. Open your application in the browser
3. You should see the feature flag content rendered
4. Go back to the Unleash dashboard and toggle the feature off/on
5. Refresh your React app to see the changes in real-time

Advanced Configuration

Using Context for Targeted Flags

Pass user context to enable targeted feature rollouts:

```
import { useFlag } from '@unleash/proxy-client-react';
import { useUnleashContext } from '@unleash/proxy-client-react';

function UserSpecificFeature() {
const { setContext } = useUnleashContext();
const isEnabled = useFlag('premiumFeature');

React.useEffect(() => {
// Set user context for targeting
setContext({
userId: 'user-123',
properties: {
plan: 'premium',
country: 'US',
},
});
}, [setContext]);

return (
<>
{isEnabled && (
Premium feature visible for premium users

)}
</>
);
}

export default UserSpecificFeature;
```

Percentage-Based Rollouts

In the Unleash dashboard, configure gradual rollouts:

1. Edit your feature flag
2. Add strategy: **Gradual Rollout**
3. Set percentage: `50` (to enable for 50% of users)
4. Save

Custom Strategies

Define complex targeting rules in the dashboard using strategy constraints:

- Enable feature for specific user IDs
- Enable based on email domain
- Enable by geographic location
- Enable based on custom properties[1]

Multiple Environments

Manage separate configurations:

1. Create feature flags with different states for each environment
2. Use environment-specific API tokens
3. Control production rollout independently from staging

Best Practices

1. Naming Conventions

Use clear, descriptive names for feature flags:

✅ Good: payment-processing-v2
❌ Bad: feature1, test_flag

2. Clean Up Old Flags

Remove feature flags that are no longer in use (100% enabled permanently) to reduce technical debt.

3. Document Your Flags

Add descriptions to feature flags explaining their purpose and target users.

4. Use Gradual Rollouts

Start with small percentages:

- 5% → 10% → 25% → 50% → 100%

This allows you to catch issues early with minimal user impact.

5. Monitor and Evaluate

Track metrics for flagged features:

- User adoption
- Error rates
- Performance impact
- User feedback

6. Test Both Paths

Ensure your application works correctly with the flag both enabled and disabled.

```
// Test this code path
if (isEnabled) {
// New feature
}
```

```
// And this code path
if (!isEnabled) {
// Old feature
}
```

7. Use Type Safety

For TypeScript projects, define types for your flags:

```
interface FeatureFlags {
newFeature: boolean;
betaFeatures: boolean;
}
```

8. Secure Your Tokens

- Never commit API tokens to version control
- Use environment variables for token storage
- Rotate tokens regularly
- Use different tokens for different environments[1]

Troubleshooting

Connection Issues

Problem: React app cannot connect to Unleash server

Solution:

1. Verify Docker containers are running: `docker ps`
2. Check Unleash server is accessible: `http://localhost:4242`
3. Verify API token is correct in your configuration
4. Check browser console for CORS errors

Feature Flag Not Updating

Problem: Feature flag changes in dashboard don't reflect in app

Solution:

1. Hard refresh browser (Ctrl+Shift+R / Cmd+Shift+R)
2. Check that the flag name matches exactly (case-sensitive)
3. Verify the flag is enabled for the correct environment

Docker Issues

Problem: Cannot start Docker containers

Solution:

1. Ensure Docker Desktop is running
2. Check available disk space
3. Stop existing containers: `docker stop unleash-server unleash-postgres`
4. Remove containers: `docker rm unleash-server unleash-postgres`
5. Restart: `docker compose up -d`

Quick Reference

Useful Docker Commands

Start Unleash

docker compose up -d

View running containers

docker ps

View logs

docker logs unleash-server

Stop Unleash

docker compose down

Stop and remove everything

docker compose down -v

Key URLs

- **Dashboard**: http://localhost:4242
- **API**: http://localhost:4242/api
- **Frontend API**: http://localhost:4242/api/frontend

Default Credentials

- **Username**: admin
- **Password**: unleash4all

Summary

Unleash enables teams to:

✅ Release features safely with gradual rollouts
✅ Test new ideas without full deployment
✅ Instantly disable problematic features
✅ Target specific user segments
✅ Maintain separation between code deployment and feature visibility

By following this guide, you now have a fully functional local Unleash setup integrated with your React application, ready for feature experimentation and controlled releases[1][3].

References

[1] Mastri, M., & Ferranti, M. (2025). Mastering Feature Flags Management with Unleash: Complete Guide. DEV Community. Retrieved from dev.to

[2] Qavi Technologies. (2024). Unleash: Integrating Feature Flags in React Applications. Retrieved from qavi.tech

[3] Einfochips. (2025). Mastering Feature Flags with Unleash: A Game-Changer for App Development. Retrieved from einfochips.com

[4] Unleash Documentation. (2025). How to Implement Feature Flags in React. Retrieved from docs.getunleash.io

[5] USAVPS. (2024). How to Install Unleash Using Docker. Retrieved from usavps.com