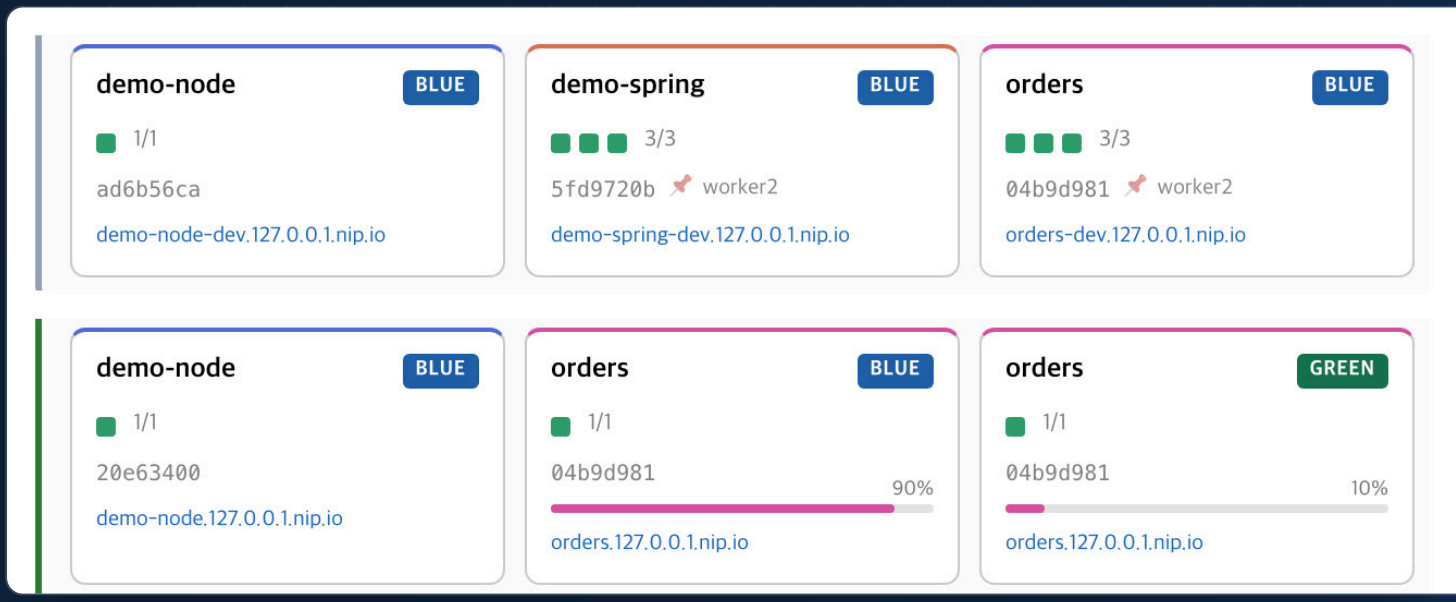
uEngine Cloud
SELF-SERVICE PaaS PORTAL

PRODUCT OVERVIEW AND ADOPTION PROPOSAL

# Developers handle the code. The portal handles Kubernetes.

Create a service from the catalog, and everything from repository creation, build and deployment to zero-downtime switchover and operations is done on one screen. uEngine Cloud is **a developer self-service PaaS portal that runs on Kubernetes**.




## 3 stages

Development · staging · production in one portal

## 5

Built-in zero-downtime deployment strategies

## One screen

Cluster-wide deployments and node placement

**October 2026**

Kubernetes-based · on-premises installation

uEngine Solutions, Inc.

CONTENTS

# Contents


01 Background 3
02 Product overview 4
03 Deliverables ① Self-service portal 5
04 Deliverables ② Deployment pipeline 6
05 Product screens at a glance 7
06 Strength ① One connected flow 8
07 Strength ② Zero-downtime deployment 9
08 Strength ③ Built-in operations tools 10
09 Live logs and container access 11
10 Strength ④ Operations you can roll back 12
11 The whole cluster on one screen 13
12 Strength ⑤ Groups and permissions 14
13 Strength ⑥ Open · on-premises 15
14 System architecture and runtime 16
15 Technology stack 17
16 Usage ① Creation to dev deployment 18
17 Usage ② Promotion and switchover 19
18 Security · access control 20
19 Installation · delivery 21
20 Verification 22
21 References 23
22 Adoption process and schedule 24


## Summary


Kubernetes is powerful, but deploying one service means handling manifests, CI settings, ingress and secrets separately. uEngine Cloud bundles this work **per app** and handles it in one portal. The logs, terminal, configuration and zero-downtime switchover you need for operations are on the same screen.


### App-level abstraction

**Developer view**

No manifests to write: set a name and resources. You can deploy a service without knowing cluster concepts

### Zero-downtime switchover

**Operations view**

Adjust blue/green, canary and A/B on screen. If something goes wrong, roll back to the previous version at once

### All in the portal

**Organization view**

Creation, build, deployment and operations form one flow. There are no separate tools to connect

**Key difference**

Most developer portal products offer only a catalog and documentation, and leave the actual build and deployment to other tools. uEngine Cloud puts the portal and the execution layer in one product, so you can go from service creation to operations without connecting several tools. The source is open, so it installs as is on in-house clusters and air-gapped networks, and its screens and guidance text are in Korean. If you already run a repository, registry or authentication server, you can connect and keep using it.

01 · BACKGROUND

# Problems that remain after adopting Kubernetes

Container orchestration is now the standard, yet developers have more concepts to handle. To deploy one service, you write a **Deployment, Service, Ingress and ConfigMap**, then set up a CI pipeline and registry authentication separately.

PROBLEM 1

### Time spent preparing deployments

Each new service repeats repository, CI, manifest and domain setup. Every team does it differently, so standards are hard to keep.

PROBLEM 2

### Requests pile up on the platform team

Even adding instances or changing a variable goes through platform staff. They handle repeated requests while developers wait.

PROBLEM 3

### Zero-downtime deployment is hard to start

Blue/green and canary deployment mean adopting and learning separate tools. Teams fall back to night-time, all-at-once releases.

PROBLEM 4

### Scattered operations tools

Logs, metrics and container access each live in a different tool. During an incident you move between many screens.

## Sticking points when evaluating developer portal tools

| Type | What to check before adopting |
|---|---|
| **Cluster management console** | Shows cluster resources, but has no flow to create and deploy services. |
| **Developer portal tools** | Catalog and docs first; builds and deployments need other tools. |
| **Commercial PaaS services** | Easy to use, but hard to install air-gapped or on your own cluster. |
| **Building it yourself** | Maintenance and handover keep needing people after the build. |
| **Deployment automation tools** | Deployment is automated; creation and operations screens are not. |

**What you need is a single flow.** Developers should be able to create a service, build it, move it up stage by stage and roll it back when something goes wrong, on their own.

02 · PRODUCT

# Introducing uEngine Cloud

uEngine Cloud is a **developer self-service PaaS portal** that runs on a Kubernetes cluster. Developers pick a template from the catalog to create a service, then push code to its repository. This starts an automated process, the **pipeline**, that takes the source, builds an image and deploys it. Operators check resources, deployment history and logs per app.

INPUT

### Templates and code

Pick a template from the catalog and set the app name and resources. After that, you only push source code to the repository.

PROCESSING

### Repository to deployment

A repository is created automatically. A container image is built, registered in the registry and deployed to your stage.

OUTPUT

### A running service

A service with its access URL ready, plus screens to check and adjust resources, logs, metrics and deployment history.

## Compared with the conventional approach

| Task | Building it yourself | uEngine Cloud |
|---|---|---|
| **New service** | Repository, CI setup, four manifests | Pick a template and enter a name |
| **Image build** | CI tools, runners, registry auth | Built in the cluster, shown in the portal |
| **Resources** | Edit the manifest, apply it again | Set instances, CPU, memory on screen |
| **Zero downtime** | Adopt and learn a separate tool | Set the share on the strategy screen |
| **Stage moves** | Rebuild, or copy images by hand | Promote the verified image as is |
| **Operations** | Separate log, metric, shell tools | Check in the app detail screen |

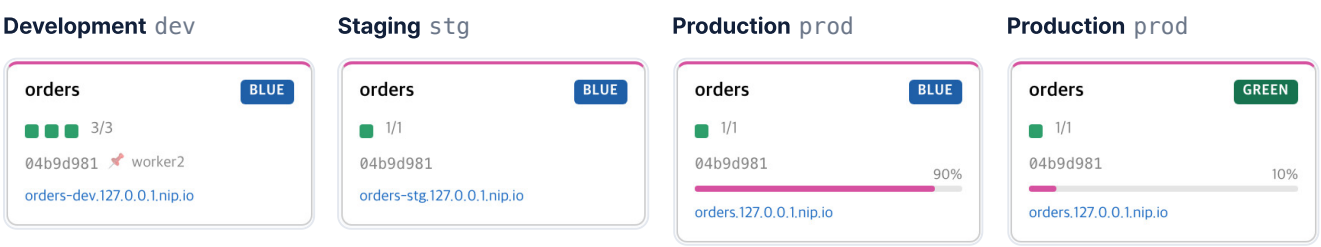



**Figure 1.** The cluster map, the portal's first screen, is divided into development, staging and production and shows every deployment. Here, the service `orders` in each zone: production splits traffic 90 percent previous (blue) and 10 percent new (green). Cards are cropped from the screen; zone names are set in English. *Demo data.*

03 · DELIVERABLES

# Deliverables ① — Self-service portal

Everything needed to run one app is organized into **10 menus**. Developers and operators look at the same screens and do different work.

| Menu | What it provides |
|---|---|
| **Getting started** | Repository clone command, access URLs per stage, what to do next |
| **Overview** | Status of development, staging and production; promotion of a verified image to the next stage |
| **Runtime & environment** | Instances, CPU and memory, environment variables, container access, configuration |
| **Source code** | Browse and read repository files inside the portal |
| **Build & deploy** | Pipeline progress, build settings, deployment strategy |
| **Snapshots** | Save deployment state and configuration; restore to a chosen point |
| **Logs** | Live log streaming, search and severity filters, save to file |
| **Monitoring** | CPU, memory and healthy-instance trends, cluster events |
| **Registry** | List of built images, redeploy of earlier versions |
| **Change history** | Records of creation, deployment, promotion, share changes and restores |

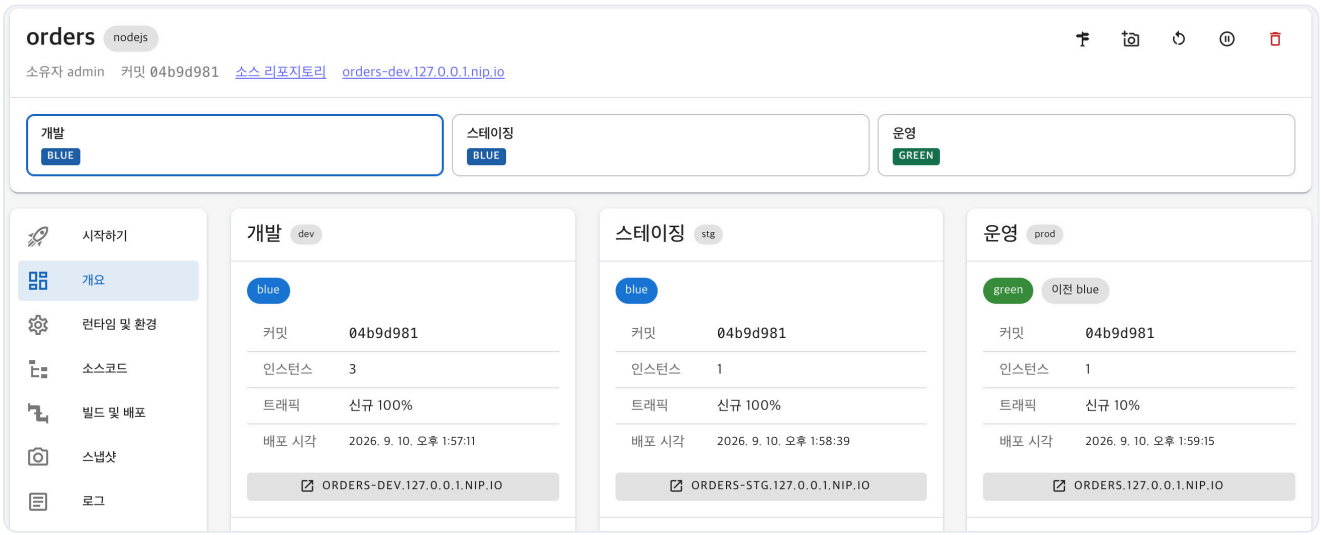



**Figure 2.** App detail overview: slots, commits and traffic share of the three stages at once. Production keeps the new slot (green) and the previous slot (blue) side by side. *Demo data · portal UI in Korean; the menu at left lists the ten menus above, in order.*

03 · DELIVERABLES

# Deliverables ② — Deployment pipeline

Push code to the repository and a build job runs inside the cluster, with no separate CI tool. The result is registered in the registry and deployed to the stage you set.

1 Source push → 2 Image build → 3 Registry → 4 Stage deployment

## Settings you adjust in the portal

| Setting | Description |
|---|---|
| **Auto trigger** | Whether the pipeline starts automatically when code is pushed. |
| **Target branches** | The branches that start it automatically. You can set more than one. |
| **Auto-deploy stage** | The stage to deploy to after a successful build. Leave it empty to build the image only. |
| **Build timeout** | When to stop a build that runs too long. |

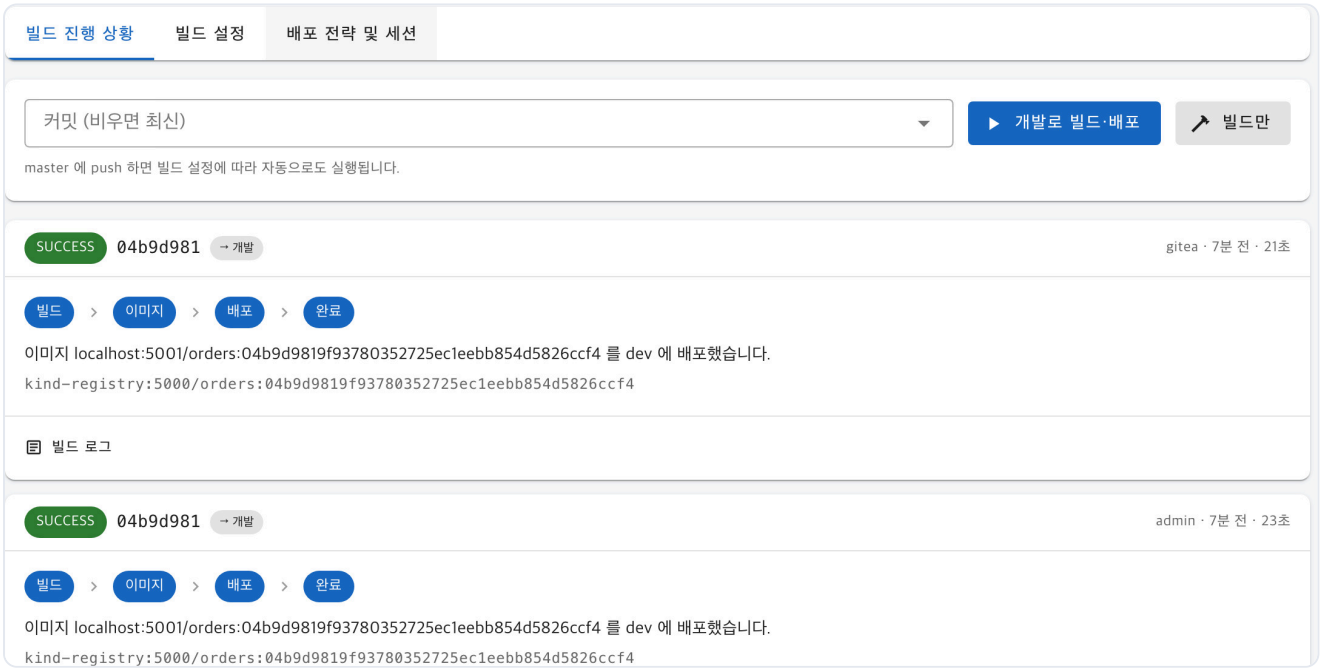



**Figure 3.** Build progress. Each step and the resulting image are shown, and the build log can be opened as is. *Demo data · portal UI in Korean (steps: build › image › deploy › done).*

> **No CI server to install.** Builds run as isolated jobs inside the cluster and are cleaned up when they finish. There are no runners to keep running.

04 · SCREENS

# Product screens at a glance

All screens below are taken from **a cluster that is actually running**.

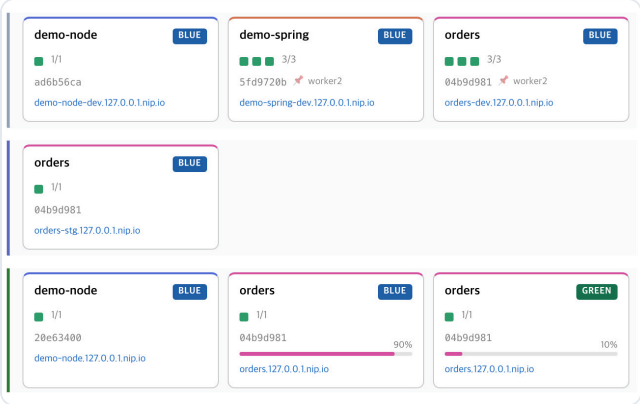



**Figure 4.** Cluster map — deployment status by zone at a glance.

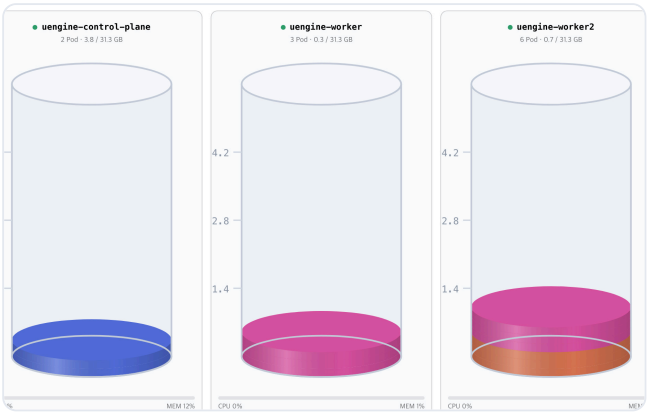



**Figure 5.** Node placement — which app sits on which node, and how much.

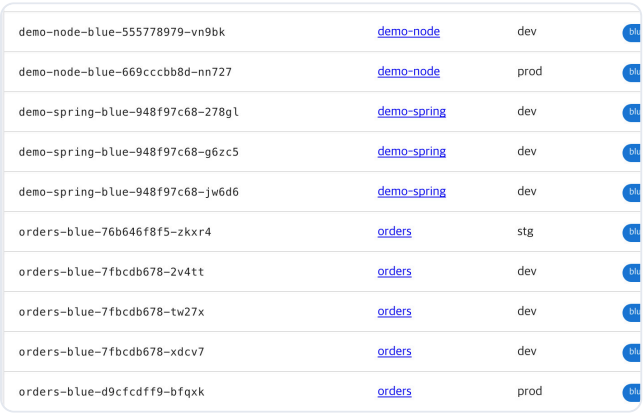



**Figure 6.** System metrics — the instances the product deployed, by stage, slot and node.




**Figure 7.** Blue/green — the demo app's page shows which slot answered.

### Screen layout

The left menu has Dashboard · Apps · App groups · Registry · System metrics · Organization. Select an app and the ten menus described earlier open.

### About the screenshots

The current portal shows its UI in Korean only; it has no language setting. This edition crops each screen to the parts that read without Korean. The few screens kept with Korean labels say so in their captions, and their menu and field names are translated in the text.

05 · STRENGTHS

# Strength ① — From service creation to operations in one flow

No chain of tools to connect. The moment you name an app in the catalog, its repository, build, deployment and access URLs are prepared together.




## What is prepared when you create an app

- **Git repository** — holds the chosen template's source and its container build file.
- **Push integration** — pushing code starts the pipeline.
- **Access URLs per stage** — for development, staging and production. You can also set them yourself.
- **Configuration files** — one per stage, created for you and passed to the container.

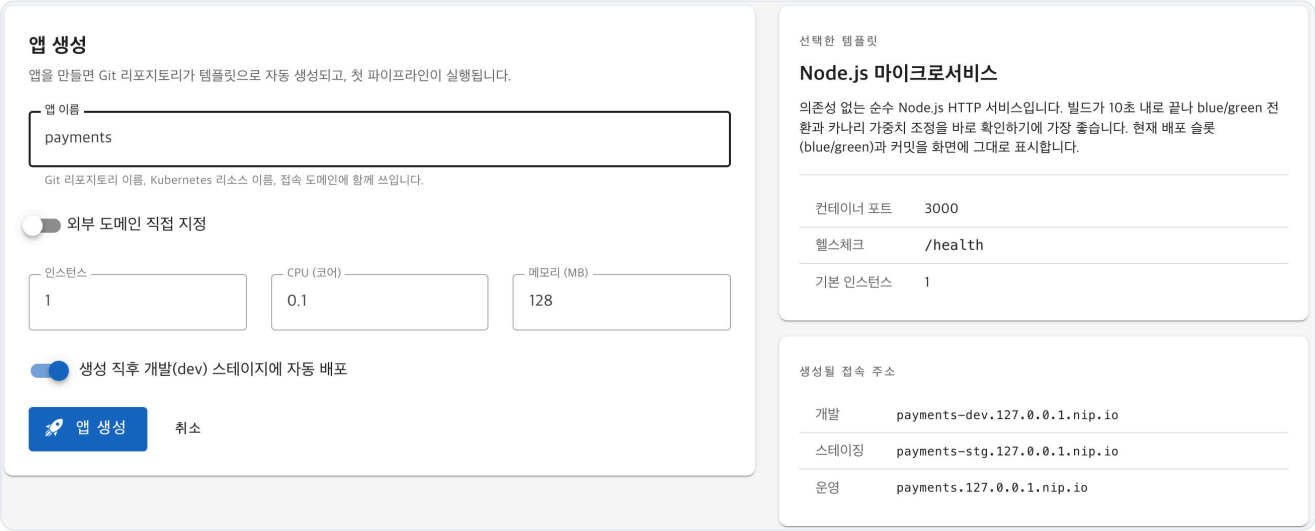



**Figure 8.** App creation screen. Enter the production domain and the staging and development URLs are set with it. *Demo data · portal UI in Korean.*

### What a template contains

Runnable sample code, a container build file, a health-check path and a per-stage configuration skeleton. The app is built and deployed right after creation, so development starts from a working state.

> Three templates are provided: **Node.js, Spring Boot 3 and Vue 3**. You can add templates that follow your organization's standards.

05 · STRENGTHS

# Strength ② — Zero-downtime deployment on screen

Production runs with **two deployment slots**. The new version goes to the idle slot, and you switch over by adjusting the traffic share. The previous slot stays, so you can go back at any time.

### Deployment strategies

| Strategy | How it works |
|---|---|
| **Blue/green** | Deploys the new version to the opposite slot and switches; the previous slot is kept for rollback. |
| **Canary** | Sets the share of traffic sent to the new version; you adjust it on screen. |
| **A/B testing** | Sends only requests with a specified request header to the new version. |
| **Session affinity** | Routes requests from the same user to the same instance. |
| **Rolling speed** | Sets the share of instances started and stopped at once during replacement. |




**Figure 9.** A request served by the previous version.




**Figure 10.** A request served by the new version.

> **Measured result** — With the new version's share set to 10 percent, 20 requests to the production URL went 16 to the previous version and 4 to the new version. We confirmed that traffic is split by the configured share.

05 · STRENGTHS

# Strength ③ — Operations tools inside the portal

You don't have to move between screens to look into a problem. Logs, metrics, container access and configuration are all **in the same app screen**.

## Configuration

Edit each stage's configuration directly in the portal's editor. Saving marks the app as needing a redeploy, and the change reaches the container on redeploy. Settings are passed as environment variables and also as a file. If the syntax is wrong, the editor tells you why when you save.

```
# orders · dev configuration
#
# Values under env are injected as container environment variables.
# The whole file is mounted in the container at /config/application.yml.
#
# Saving shows a "Redeploy required" mark; changes apply on redeploy.

env:
  LOG_LEVEL: INFO
  GREETING: "Hello, this is orders."
```

**Figure 11.** The configuration the portal creates for each stage, here for the demo app `orders` in development. Redrawn from the configuration editor with the comments and the greeting translated. *Demo data.*

## Resources

Change the instance count, CPU and memory with the controls on screen. Before you save, changed values are shown next to the current ones, so you can see what will change before applying it.




| | | | | | |
|---|---|---|---|---|---|
| orders-blue-7fbcdb678-2v4tt | uengine-worker2 | Ready | 0 | 0m | 10 MB |
| orders-blue-7fbcdb678-tw27x | uengine-worker2 | Ready | 0 | 0m | 10 MB |
| orders-blue-7fbcdb678-xdcv7 | uengine-worker2 | Ready | 0 | 0m | 10 MB |

**Figure 12.** Resource controls: instances, CPU (cores per pod) and memory (MB per pod), each adjusted with – and +. Below, the three pods now running (pod, node, status, restarts, CPU, memory). *Demo data; Korean labels, header row and age column cropped.*

05 · STRENGTHS

# Live logs and container access

The portal includes two ways to look into a problem: a screen that streams logs live, and a screen that connects straight to a container.

## Logs

- **Live streaming** — select an instance and its log keeps flowing in.
- **Search and highlight** — the text you search for is highlighted on screen.
- **Severity** — errors, warnings and info messages are color-coded, and you can filter by type.
- **Save to file** — download what you are looking at as a file.

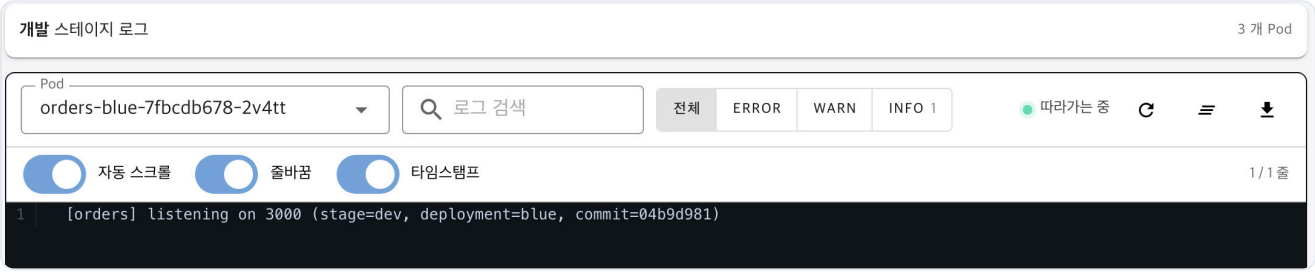



**Figure 13.** Log screen. Auto-scroll, line wrap and timestamps can each be switched on and off. *Demo data · portal UI in Korean (filters: All · ERROR · WARN · INFO; toggles: auto-scroll, line wrap, timestamps).*

## Container access

Connect to a running container straight from the browser. Instead of a separate access server, it uses the exec function that Kubernetes provides, so each session is recorded in the cluster's audit log.

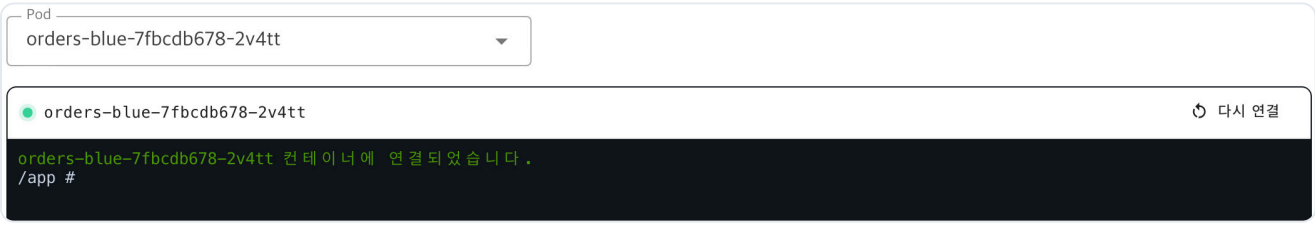



**Figure 14.** Container access screen. *Demo data · portal UI in Korean (the first line reads "Connected to container orders-blue-7fbcdb678-2v4tt.").*




**Figure 15.** Monitoring screen, which shows usage trends per app together with cluster events; this crop shows the recent cluster events. *Demo data.*

05 · STRENGTHS

# Strength ④ — Operations you can roll back

You can deploy with confidence only when you can roll back. uEngine Cloud provides **three ways to go back**.

METHOD 1

### Slot rollback

The previous slot is kept, so once a problem is confirmed you return to the previous version at once.

METHOD 2

### Image redeploy

Pick a past image from the registry and deploy it, without building it again.

METHOD 3

### Snapshot restore

Save the deployment state and configuration together, and return to the setup of a chosen point.

## What a snapshot holds

| Item | Description |
|---|---|
| **Deployed images and commits** | Records which version was deployed in each stage. |
| **Resource settings** | Saves the instance count and the CPU and memory values. |
| **Deployment strategy** | Saves the switchover method and the traffic share. |
| **Configuration** | Saves each stage's configuration. |

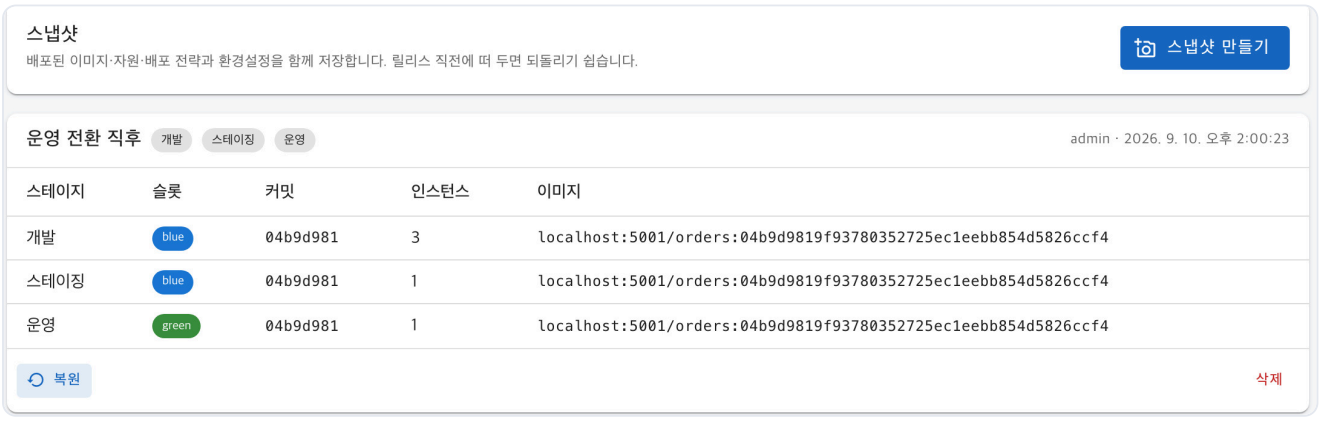



**Figure 16.** Snapshot list. Choose the stages to restore and whether to redeploy. *Demo data · portal UI in Korean (this snapshot is named "Right after the production switch").*

> Every operation is kept in the **change history**. For each app you can see who deployed or rolled back which stage, and when.

05 · STRENGTHS

# The whole cluster on one screen

The first thing you see in the portal is the **cluster map**: which services run in which stage, how many instances each has and how traffic is split, all without leaving the screen.

## Deployment status by zone

- **A zone per stage** — development, staging and production are shown as separate areas.
- **One mark per instance** — each small square is one running instance; it turns green when healthy.
- **Switchover state** — a workload being deployed has a blinking border. During a zero-downtime switchover, the new and previous versions' shares are shown too.

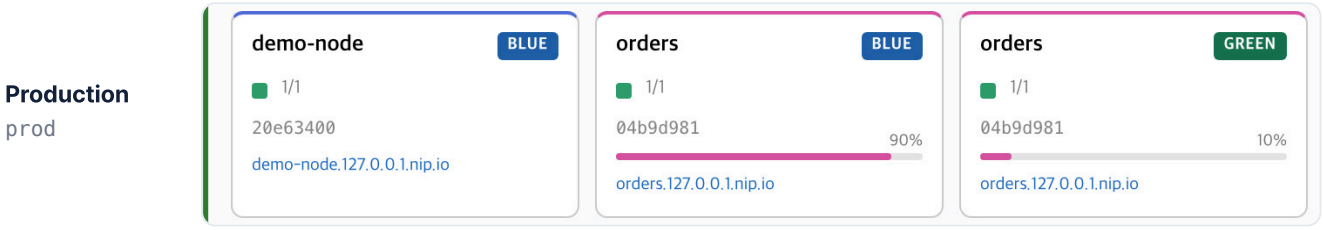



**Figure 17.** Cluster map, production zone. The service `orders` is split 10 percent new (green) and 90 percent previous (blue). *Demo data.*

## Node placement and moves

Below the map, nodes are drawn as cylinders. Each colored band is the memory a service uses on that node, and all cylinders share one scale, so you see at once which node is fuller. Click a band to **move that service to another node**.

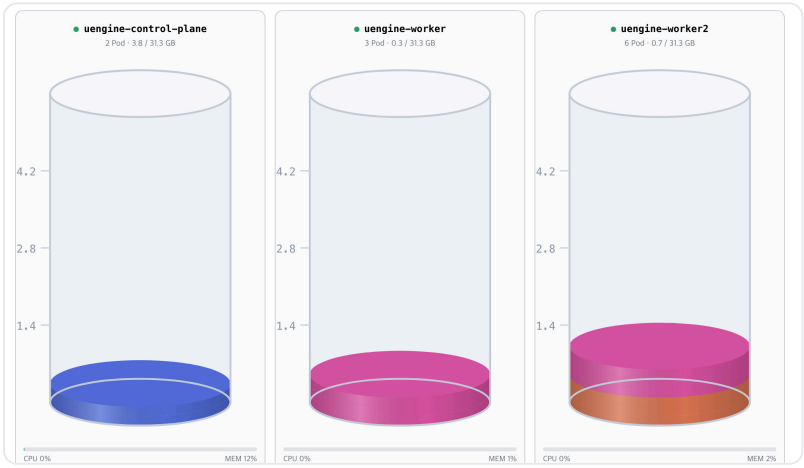



**Figure 18.** Node placement: how services are spread across three nodes. *Demo data; the buttons under each cylinder are cropped.*

> **Node-level actions** — before maintenance, block new placements on a node, or move the services on it to other nodes.

05 · STRENGTHS

# Strength ⑤ — Managing several services as a group

Microservices don't move alone. Put services that must be deployed together into an **app group**, and promote them or take snapshots in one step.

## App groups

- **Group promotion** — moves the group's services together from development to staging, and from staging to production.
- **Results** — shows success or failure for each service. If some fail, the rest still go ahead.
- **Group snapshot** — saves the whole group as snapshots under one name.
- **Status by stage** — shows how many of the group's services are deployed in each stage.




**Figure 19.** App group screen. *Demo data · portal UI in Korean (the group "Demo bundle": columns are app, development, staging, production).*

## Users and permissions

Portal users are managed in the Organization menu. Add and deactivate accounts, reset passwords, assign roles and set group membership on screen. Authentication is handled by an authentication server that follows standard protocols, and it can be linked to your in-house account system.

| | | |
|---|---|---|
| admin | Cloud Admin | admin@uengine.org |
| developer | Dev User | developer@uengine.org |

**Figure 20.** Organization screen, user list (ID, name, email). *Demo data; header row and status columns cropped.*

05 · STRENGTHS

# Strength ⑥ — Open · on-premises · Korean

The source is open and nothing depends on external services. It installs as is on in-house clusters and air-gapped networks.

OPEN

### MIT-licensed open source

Source and configuration are public. Extend it, and adapt templates and screens to your standards.

ON-PREMISES

### No outside connections

Portal, repository, registry and authentication all run in the cluster. No sign-up, no usage fees.

KOREAN

### Screens and docs in Korean

Menus, guidance and error messages are in Korean, so teams in Korea can run it without a handover.

STANDARDS

### Standard Kubernetes resources

No proprietary formats. It works alongside your existing cluster tools.

## Working with your existing cluster tools

| Tool you already use | How they work together |
|---|---|
| **Cluster management console** | Standard resources, so deployments also show in your console. |
| **Monitoring · log collection** | Keep your existing collection setup next to the portal's screens. |
| **In-house authentication** | Standard protocols, so it links to your in-house account system. |
| **In-house repository · registry** | At installation, point it to your existing repository and registry. |

## Extension points

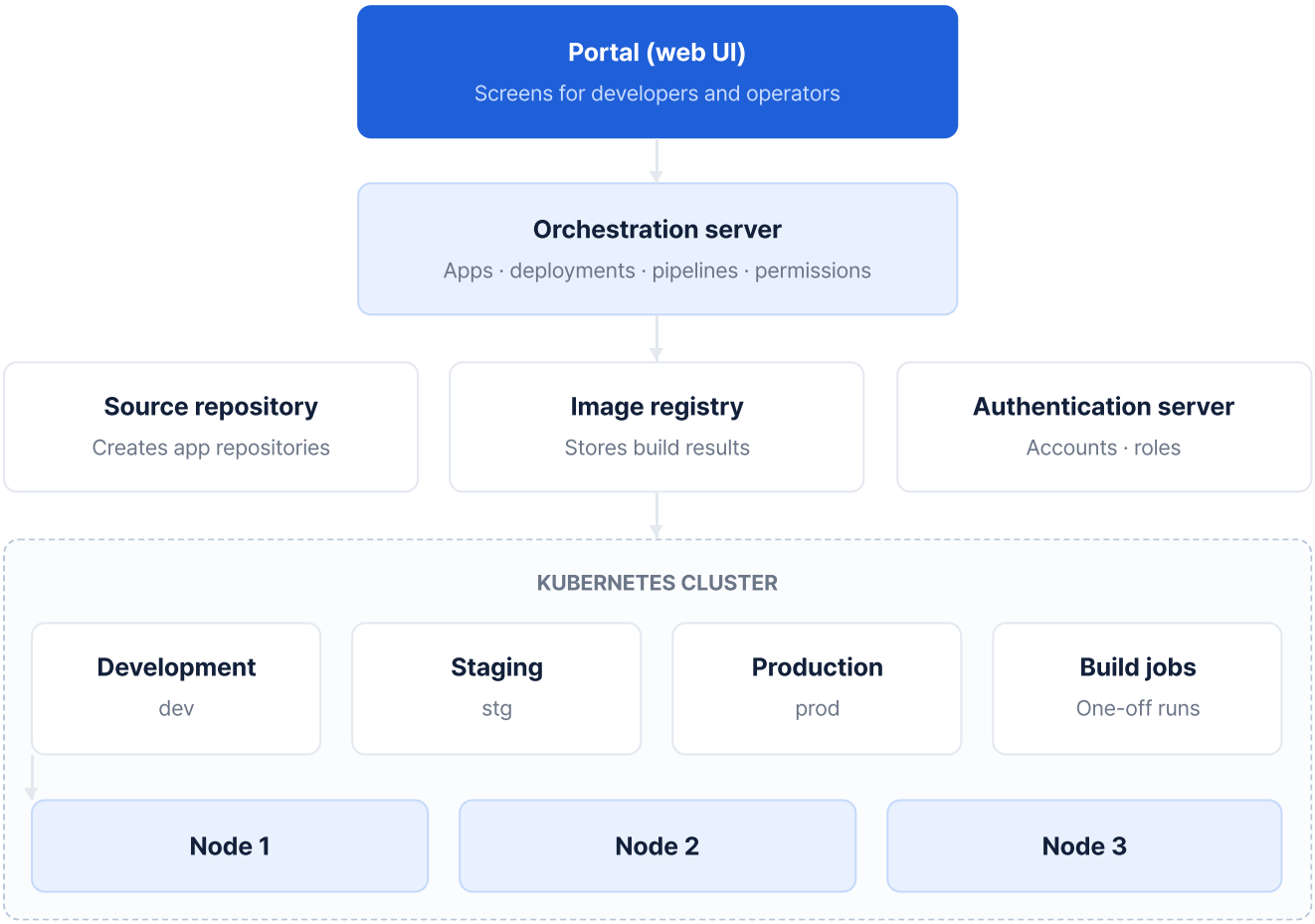
| What to extend | How |
|---|---|
| **App templates** | Add templates with your standard project structure and build method. |
| **Deployment defaults** | Adjust default resources per stage, health-check path, switchover method. |
| **Portal screens** | Adapt layout and wording to your internal terms and procedures. |

**No technology lock-in.** The source is open, so your organization can maintain and extend it directly.

06 · ARCHITECTURE

# System architecture and runtime

The product consists of the portal, an orchestration server and supporting components that run in the cluster. **Every component runs inside a single Kubernetes cluster.**




## Stage separation and node placement

Development, staging and production are deployed to separate areas, and each stage keeps its own resources and configuration. The scheduler picks the node; you can specify it in the portal when needed.

## Runtime requirements

| Item | Requirement |
|---|---|
| **Kubernetes** | 1.28 or later. Both managed services and self-built clusters are supported. |
| **Ingress controller** | A controller that supports splitting traffic by weight is required. |
| **Storage · metrics** | Repository and registry capacity is sized by the number of services. A metrics component is required to show usage. |

06 · ARCHITECTURE

# Technology stack

Current versions, tied to no commercial product: each layer can be replaced with your standard product.

| Layer | Technology | Role |
|---|---|---|
| **Portal UI** | Vue 3.5 · Vuetify 3.7 · Vite 6 | Screens for developers and operators |
| **Orchestration** | Spring Boot 3.5 · Java 17 | Apps, deployments, pipelines, permissions |
| **Cluster link** | Fabric8 Kubernetes Client 7.9 | Creates standard resources, reads status |
| **Image build** | BuildKit | Builds images inside the cluster |
| **Repository** | Gitea | App repositories, push integration |
| **Authentication** | Keycloak (OAuth 2.0 · OIDC) | Login, roles and groups |
| **Traffic routing** | Ingress NGINX | Access URLs, switchover shares |

## Design principles

### Standard resources only

No proprietary formats. Results also show in your existing cluster tools.

### App-level abstraction

Everything is organized per app. Developers don't handle cluster concepts.

### Replaceable components

Repository, registry and authentication are settings; swap in your own products.

## Versions and maintenance

| Item | Details |
|---|---|
| **Base versions** | Current stable release for each layer; you can pin a long-term support version. |
| **Cluster compatibility** | Standard resources only, so cluster upgrades have little impact. |
| **Replacing components** | Repository, registry and authentication are settings; switch to in-house products. |

### Product size

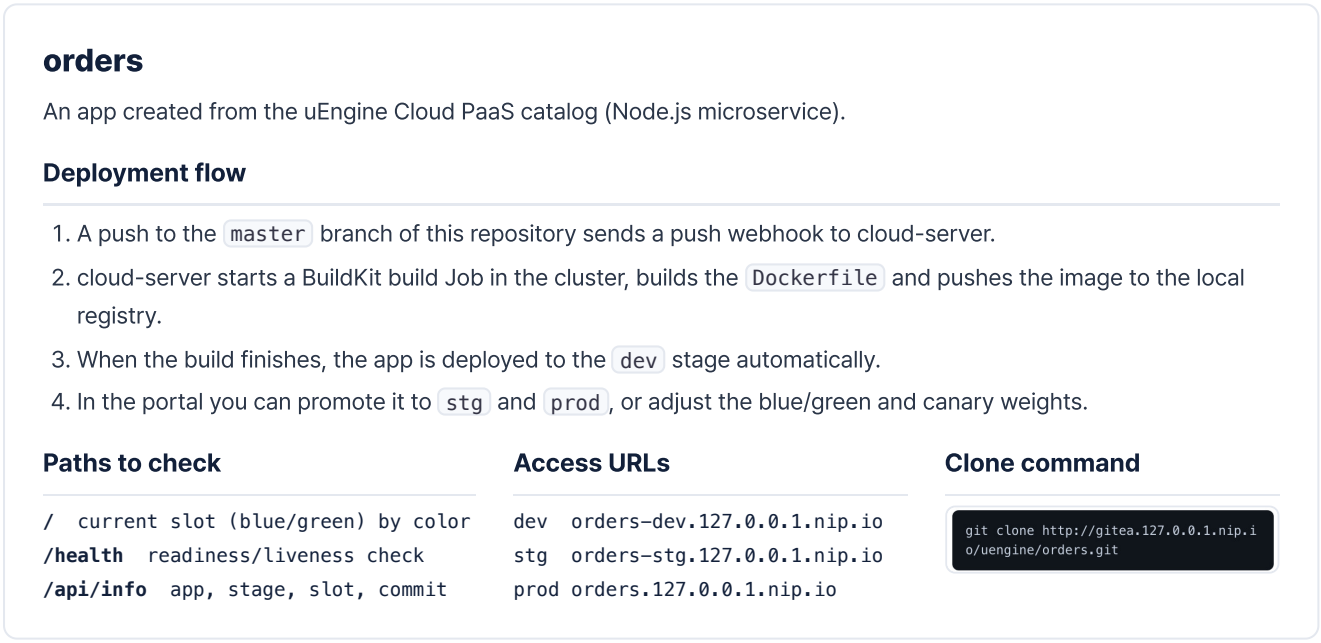
The orchestration server provides **76 functional interfaces** across 12 areas, and the portal consists of **27 routes**.

07 · HOW TO USE

# Usage ① — From service creation to development deployment

These steps were performed on a real cluster. With the Node.js template the whole process finishes **within 5 minutes**; templates with longer builds take correspondingly longer.

1. **Pick a template from the catalog** — choose the type of service to create.
2. **Set the name and resources** — app name, instances, CPU and memory; the access URLs are shown in advance.
3. **Create** — the repository is created and the first pipeline starts.
4. **Check the build** — watch the progress and the build log on screen.
5. **Deployed to development** — when the build ends, the app runs in development and its URL opens.



### orders

An app created from the uEngine Cloud PaaS catalog (Node.js microservice).

#### Deployment flow

1. A push to the `master` branch of this repository sends a push webhook to cloud-server.
2. cloud-server starts a BuildKit build Job in the cluster, builds the `Dockerfile` and pushes the image to the local registry.
3. When the build finishes, the app is deployed to the `dev` stage automatically.
4. In the portal you can promote it to `stg` and `prod`, or adjust the blue/green and canary weights.

#### Paths to check

```
/  current slot (blue/green) by color
/health  readiness/liveness check
/api/info  app, stage, slot, commit
```

#### Access URLs

```
dev  orders-dev.127.0.0.1.nip.io
stg  orders-stg.127.0.0.1.nip.io
prod orders.127.0.0.1.nip.io
```

#### Clone command

```
git clone http://gitea.127.0.0.1.nip.io/uengine/orders.git
```

**Figure 21.** The Getting started page shows the app's README, the clone command and next steps. Shown: the README the Node.js template writes, redrawn in English (path notes shortened), and the clone command cropped from the screen. *Demo data.*

## The development flow after that

Developers clone the repository, change the code and push it. When code reaches the configured branch, the pipeline starts and the change reaches development. There is no separate deployment request. If a build fails, the pipeline screen shows the failed step and its build log; fix the code and push again.

> In the build settings you can adjust **auto trigger and target branches**. For a review before deployment, leave the auto-deploy stage empty and deploy from the portal.

07 · HOW TO USE

# Usage ② — Promotion and zero-downtime switchover

The image verified in development moves up to the next stage **without being rebuilt**. The same image goes up, so the stages don't drift apart.




## Production switchover steps

1. **Deploy the new version** — it goes to the idle slot. The current slot keeps serving.
2. **Set the share** — choose the share of traffic sent to the new version.
3. **Check** — verify the new version's behavior with logs and metrics.
4. **Confirm or roll back** — if all is well, raise the share and confirm; if not, go back to the previous version.

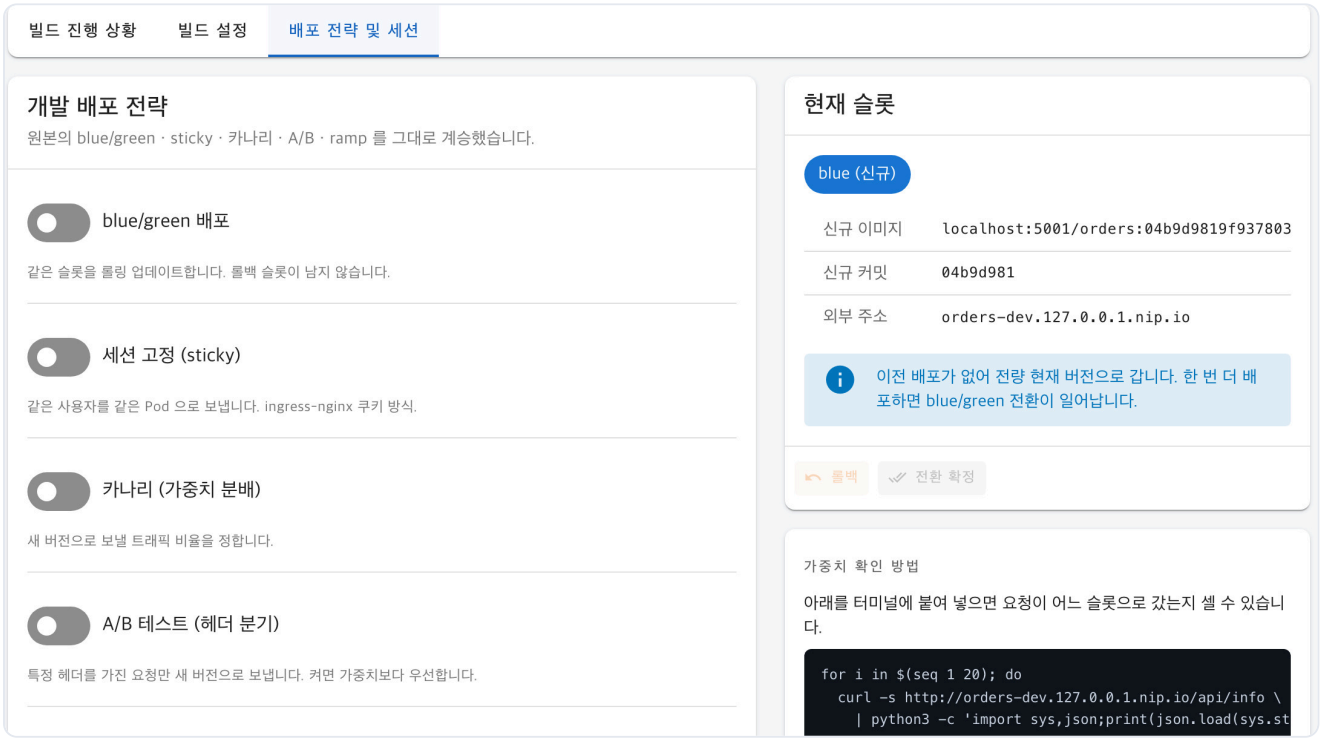



**Figure 22.** Deployment strategy screen, with the current slot setup and a way to check the split. *Demo data · portal UI in Korean (left: blue/green, session affinity, canary, A/B testing, rolling speed; right: current slot and a* `curl` *loop that counts which slot answered).*

> **The previous version stays until you confirm the switch.** Confirming removes the previous slot and sends every request to the new version. Even after that, you can redeploy the earlier image kept in the registry.

08 · SECURITY

# Security · access control

Authentication is handled by an authentication server that follows standard protocols. Cluster operations are performed only within the permissions granted to the orchestration server.

## Authentication and permissions

| Item | Details |
|---|---|
| **Login** | OAuth 2.0 · OIDC; can be linked to your in-house account system. |
| **Roles** | Administrator or regular user, assigned in the Organization menu. |
| **Groups** | Users are grouped by department or team. |
| **Token validation** | The server validates the token of every portal request before processing it. |

## Records and tracking

- **Operation history** — creation, deployment, promotion, share changes, restores and deletions, per app.
- **Container access** — sessions use Kubernetes exec, so they are recorded in the cluster audit log.
- **Builds** — which commit produced which image is recorded and shown in the image list.

## Air-gapped environments

The portal, repository, registry and authentication server all run in the cluster, so it operates without external communication. For an air-gapped network, the required container images are brought in before installation.




**Figure 23.** Change history screen. Time, status, action and actor are recorded together. *Demo data · portal UI in Korean (the action column shows operation types such as UPDATE_APP and CREATE_APP_SNAPSHOT).*

> **To agree before adoption** — how to link your in-house authentication, how images are brought in, and the scope of cluster permissions are decided together at the start of the build.

09 · DELIVERY

# Installation · delivery

Installation starts with a single script. Everything from cluster preparation to portal access is installed together.

## Scope of delivery

| Item | Contents |
|---|---|
| **Product source** | Full source and build configuration of the portal and the orchestration server |
| **Installation setup** | Cluster configuration files, definitions for the supporting components, installation script |
| **App templates** | Node.js · Spring Boot 3 · Vue 3 templates. More can be added to match your organization's standards |
| **Documentation** | Installation guide, concept mapping table, a description of each screen, demo scenario |
| **Demo video** | The whole process from service creation to zero-downtime switchover and node placement (about 6 minutes, with subtitles) |

## Installation steps

1. **Prepare the cluster** — use an existing cluster, or create one from the configuration files provided.
2. **Install supporting components** — install the repository, authentication server, registry and traffic routing components.
3. **Start the server** — start the orchestration server and connect its cluster access permissions.
4. **Open the portal** — log in with the administrator account and create the first app.

### Using what you already have

If you already run a repository or image registry, give its address at installation and keep using it. There is no need to install these components twice.

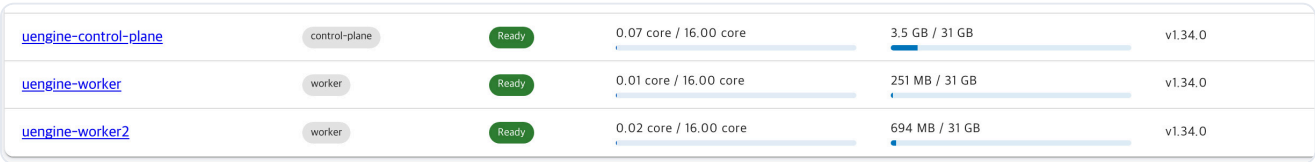



**Figure 24.** System metrics screen, node status: name, role, status, CPU, memory and kubelet version of the three nodes. The same screen lists the instances the product deployed (next page). *Demo data; table header cropped.*

10 · VERIFICATION

# Verification

The items below are **results confirmed on a real Kubernetes cluster**. We checked not only how the screens behave but also what was applied to the cluster.

| Item | How it was checked, and the result |
|---|---|
| **Service creation** | App created → repository, template contents and push integration confirmed |
| **Image build** | Build job ran in the cluster → image registered in the registry |
| **Stage deployment** | Development auto-deploy → staging → production; the same image in all three stages |
| **Configuration** | Values saved in the portal reached the running container's environment and config file |
| **Resources** | Instance count changed → the actual number of instances changed |
| **Stop · resume** | 0 instances when stopped; the previous count back on resume |
| **Zero downtime** | New version at 10 percent, 20 requests → 16 to the previous, 4 to the new version |
| **Session affinity** | When set, the session-affinity entry was applied to the traffic routing configuration |
| **Live logs** | Logs kept streaming in the portal |
| **Container access** | Opened a container shell in the browser, ran commands and checked the results |
| **Snapshots** | After saving, images, commits and instance counts per stage were listed |
| **Group promotion** | Two services grouped; deployment status counted per stage |
| **Node move** | Node chosen in the portal → instance restarted there, band moved (3-node cluster) |



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| demo-node-blue-555778979-vn9bk | demo-node | dev | blue | uengine-control-plane | Running | 0m | 13 MB |
| demo-node-blue-669cccbb8d-nn727 | demo-node | prod | blue | uengine-control-plane | Running | 0m | 12 MB |
| demo-spring-blue-948f97c68-278gl | demo-spring | dev | blue | uengine-worker2 | Running | 1m | 147 MB |
| demo-spring-blue-948f97c68-g6zc5 | demo-spring | dev | blue | uengine-worker2 | Running | 1m | 145 MB |
| demo-spring-blue-948f97c68-jw6d6 | demo-spring | dev | blue | uengine-worker2 | Running | 2m | 144 MB |

**Figure 25.** System metrics, the instances the product deployed (pod, app, stage, slot, node, status). *Demo data; header cropped.*

> **Demo video** — a screen recording of about 6 minutes that performs the steps above in sequence is provided.

11 · REFERENCES

# References

Major uEngine Solutions projects in **PaaS, microservice transition and business process management**.

| Customer | Key work |
|---|---|
| **KIAT · public R&D institution** | OCE-based cloud pool and shared app store (2013–2014) |
| **Hanwha Group · Hanwha Systems** | BPM adoption, microservice consulting and training (2003–2025) |
| **Gov24** | Cloud-native microservice transition consulting (2025) |
| **POSCO DX · POSCO** | Logistics consulting, AI-based design and modernization (2022–2025) |
| **SK C&C · SK Telecom** | Microservice training and operations automation (2020–2026) |
| **KB Kookmin Bank** | KB Wallet microservice implementation consulting (2023–2024) |
| **LG CNS · LG Energy Solution** | Microservice methodology; manufacturing and ERP BPM (2019–2024) |

## Product lineage

**2003** uEngine BPM released → **2013** OCE open-source PaaS → **2018** Microservice DevOps → **Today** Kubernetes support

The current product replaces the orchestration layer of the 2018 open-source release with Kubernetes and rebuilds the portal and server on current technology.

## Assets provided with it

| Item | Contents |
|---|---|
| **Course material** | 19 hands-on documents: decomposition, API gateway, service calls, domains |
| **Transition consulting** | Cloud-native transition assessment and architecture design support |
| **Training courses** | Courses on microservice design and implementation |

**Link to the design tool** — Deploy the outputs of Robo Architect (requirements analysis through microservice design and implementation) to uEngine Cloud, and connect design to operations in one flow.

12 · ADOPTION

# Adoption process and schedule

The standard schedule is **8 weeks**; reusing your cluster and in-house repository can shorten it.




## Deliverables by phase

| Phase | Deliverables |
|---|---|
| **Assessment** | Environment check results, integration decision document |
| **Install and integrate** | Installed environment, configuration specification, accounts and permissions |
| **Template setup** | Standard templates for your organization, build setting guidelines |
| **Pilot transition** | Transitioned services, deployment procedure definition |
| **Training, handover** | Training materials, operations guide |

## Support after adoption

### Operations support

Installation checks, incident analysis and upgrades, per contract scope.

### More templates

We write templates for each new language or framework standard.

### Transition consulting

We work with you from the design of splitting services into microservices.

**Before you decide** — during evaluation we will schedule a demo for you. A pilot installation on your in-house cluster, checked with real services, is also possible.

uEngine Cloud
SELF-SERVICE PaaS PORTAL

CONTACT US

# See it in a demo first.

A video of **about 6 minutes** covers the whole process, from service creation to zero-downtime switchover and node placement. We can also schedule a pilot installation on your in-house cluster to check it with real services.

---

**+82-2-567-8301**
Main phone

**help@uengine.org**
Adoption inquiries

**uengine-oss**
Product source on GitHub

**uEngine Solutions, Inc.**
HQ: #301, 3F Gunghyun Bldg., 31 Saimdang-ro, Seocho-gu, Seoul, Korea · Branch: 37 Sangnam-ro, Seongsan-gu, Changwon-si, Gyeongsangnam-do, Korea
© 2026 uEngine. Document Rev. 2026.10 · Figures and screens are as of the time of writing.

uEngine