CSC Rahti

CSC Rahti is a **container cloud service** provided by CSC – IT Center for Science in Finland. It allows researchers, students, and other eligible users to deploy and run containerized applications easily using Kubernetes/OKD technology.

Here's a clear overview:

🚀 What CSC Rahti Is

CSC Rahti is a **shared general-purpose container platform** for hosting applications, based on Kubernetes/OKD (the open-source version of Red Hat OpenShift). It offers an environment to run web applications, APIs, databases, scientific software stacks, and other container-based workloads. [rahti.csc.fi]

🔧 Key Features

- **Built on OKD/Kubernetes** → supports modern container orchestration. [csc-guide-...ahtiapp.fi]
- **Scalable & fault-tolerant apps** → automatic scaling, health checks, rolling updates. [docs.csc.fi]
- **Load balancing & high availability** → integrated web routing and HA features. [research.csc.fi]
- **No root-level access** → runs containers as non-privileged users for security. [rahti.csc.fi], [csc-guide-...ahtiapp.fi]
- **Default DNS & certificates** → apps get *.rahtiapp.fi domain names with HTTPS automatically included. [rahti.csc.fi], [docs.csc.fi]
- **Supports many use cases**, such as:
    - Hosting websites or APIs
    - Deploying scientific analysis tools
    - Running data pipelines
    - Packaging complex apps (e.g., Spark) for others to use [csc-guide-...ahtiapp.fi]

🌐 How It Works

In Rahti, you manage **applications**, not virtual machines. Think of it as a shared "big computer" where you deploy apps, compared to CSC's cPouta, which is more like managing your own virtual machines in a data center. [csc-guide-...ahtiapp.fi]

You can interact with Rahti using:

- A graphical **web UI**,
- Command-line tools like **oc** (OpenShift client) or **kubectl**. [research.csc.fi]

🔒 Data & Security

- Content is **stored in Finland**.
- No automatic backups — **users must handle their own backups**.
- Security restrictions are applied due to shared infrastructure (no root or privileged containers). [rahti.csc.fi]

🎓 Who Can Use It?

Rahti is **free of charge** for:

- Finnish universities and universities of applied sciences,
- Research institutes,
- Academy of Finland–funded research projects.
  [rahti.csc.fi]

Using Rahti requires:

- A CSC user account
- A CSC project
  [research.csc.fi]

📌 Summary

CSC Rahti = a secure, scalable, Kubernetes-based container cloud for Finnish research and education.
It's ideal for deploying web apps, scientific tools, and containerized workflows without managing infrastructure.

Hint: Even if Kubernets / OKD / CSC Rahti nomore use docker to run containers, it definitely still makes sense to create Dockerfile and local docker-compose files. As those are easier to understand and manage. If and when you are able to run your projects dockerized (with secrets, env variables, etc. configured) you have proof that you know what is needed for the (more complicated and harder to debug) deployment to CSC Rahti Kubernetes implementation!

Kubernetes

Kubernetes (also known as K8s) is an open-source container orchestration platform for automating the deployment, scaling, and management of containerized applications. Here's a comprehensive overview:

What is Kubernetes?

- **Definition**: An open-source system originally developed by Google (first released in 2014) and now maintained by the Cloud Native Computing Foundation, designed to manage containerized workloads across clusters of machines. [en.wikipedia.org], [kubernetes.io]
- **Name Origin**: Derived from the Greek word *kubernḗtēs* ("helmsman" or "pilot"), symbolizing its role in steering containerized applications. [en.wikipedia.org], [kubernetes.io]

Core Features & Benefits

- **Automation**: Handles deployment, scaling, rolling updates and rollbacks, and self-healing of containers. [kubernetes.io], [kubernetes.io], [kubernetes.io]
- **Service Discovery & Load Balancing**: Automatically exposes containers via DNS/IP and load-balances traffic when necessary. [kubernetes.io], [kubernetes.io]
- **Storage Orchestration**: Supports diverse storage backends (e.g., cloud, network, local) and automates volume mounting. [kubernetes.io], [kubernetes.io]
- **Self-Healing**: Restarts or replaces failed containers, and reschedules Pods if nodes die. [kubernetes.io], [kubernetes.io]
- **Secrets & Config Management**: Securely manages sensitive data and configuration without exposing it. [kubernetes.io], [kubernetes.io]
- **Bin Packing & Scaling**: Optimizes resource usage by scheduling containers based on CPU/memory; supports autoscaling. [kubernetes.io], [kubernetes.io]
- **Consistent Deployments**: Desired-state configuration ensures environments remain consistent across on-prem, cloud, or hybrid. [azure.microsoft.com], [kubernetes.io]

Architecture Overview

Control Plane (Master)

- **kube-apiserver**: Main API endpoint.
- **etcd**: Distributed key-value store for cluster state.
- **kube-scheduler**: Assigns Pods to nodes.
- **kube-controller-manager**: Ensures cluster state matches desired state.
- **cloud-controller-manager** (optional): Interfaces with cloud-specific services. [bing.com], [kubernetes.io]

Worker Nodes

- **kubelet**: Ensures Pod containers are running and healthy.
- **kube-proxy**: Manages networking and load balancing rules.
- **Container Runtime**: Executes containers (e.g., **containerd**, CRI-O). [kubernetes.io], [en.wikipedia.org]

Add-ons & Extensibility

- **DNS**, **dashboard UI**, **monitoring**, **logging** and more can be added as needed. [kubernetes.io]

Ecosystem & Use Cases

- **Broad adoption** in e-commerce, media, fintech, healthcare, CI/CD pipelines, hybrid/multi-cloud environments). [bing.com], [bing.com]
- Integrated with diverse tools (Helm, Istio, Knative) to support logging, monitoring, service mesh, serverless capabilities. [bing.com], [bing.com]

Latest Version (as of Feb 2026)

- **Kubernetes v1.35.1**: Current latest minor release, with patch release 1.35.1 dated 10 Feb 2026. [bing.com], [bing.com]
- Support is active for versions 1.33, 1.34, 1.35; older versions like 1.32 are nearing end-of-life. [bing.com], [bing.com]
- **v1.36** is currently in alpha phase, with features being frozen for its final release in April 2026. [bing.com], [bing.com]

Summary

Kubernetes empowers organizations by automating the lifecycle of containerized apps—scaling efficiently, ensuring high availability, and promoting infrastructure consistency across environments. Its strong architecture and extensibility make it the cornerstone of modern cloud-native infrastructure.

https://www.youtube.com/watch?v=oGPjzCBZGzg "***Docker vs. Kubernetes: The ONLY Video You Need to Finally Understand Containers!***"

Kubernetes Basics

cluster is the whole Kubernetes system containing the control plane and multiple worker nodes. It manages scheduling, scaling, and the overall state of applications. [bing.com], [redhat.com]
node is a single machine (physical or virtual) inside the cluster. It runs Pods and includes components like the kubelet, kube-proxy, and container runtime. [bing.com], [redhat.com]
pod is the smallest deployable unit in Kubernetes. It contains one or more containers that share networking and storage. Pods are ephemeral—and Kubernetes quickly recreates them if they crash. [bing.com], [bing.com]

Cluster → Nodes → Pods → Containers
A *cluster* consists of *nodes*.
A *node* runs *pods*.
A *pod* runs *containers*. (BTW. Nowadays containers are no more run as docker containers, but similar)

OKD | Red Hat OpenShift

OKD is the *community-driven, open-source distribution of Kubernetes* that forms the upstream foundation for **Red Hat OpenShift**. It provides a complete Kubernetes platform designed for continuous application development, multi-tenant environments, and automated cluster operations. [bing.com], [kubernetes.dev]

Key Characteristics of OKD

1. 100% Open Source

OKD is released under the Apache 2.0 license and is freely available for anyone to use, modify, or contribute to. [bing.com]

2. Built on Kubernetes

At its core, OKD *is* Kubernetes, but it extends it with additional tools and features that improve developer productivity and cluster management. This includes:

- Automated upgrades
- Integrated monitoring
- Operator-based lifecycle management
- Built-in image registry
- Web console and developer tooling (such as Source-to-Image) [bing.com], [bing.com]

3. Upstream for Red Hat OpenShift

OKD is where new features are developed and tested **before** they make their way into Red Hat OpenShift. Because of this:

- OKD is often a few releases ahead of OpenShift
- It uses **CentOS Stream CoreOS** as its underlying node OS [kubernetes.dev]

4. Multi-Cloud and Multi-Environment Support

OKD can run:

- On public clouds (e.g., AWS)
- On-premises hardware
- In virtualization environments
- On edge devices

The installer supports fully automated deployments on some platforms. [bing.com]

5. Security-Focused

OKD enforces strong default security practices such as:

- Containers running as non-root
- Strict namespace isolation
  [bing.com]

OKD vs. Red Hat OpenShift (Simple Comparison)

OKD

- Community-supported
- Free and open-source
- Uses CentOS Stream CoreOS
- Great for developers, students, labs, and testing environments
- Faster access to new features [bing.com], [kubernetes.dev]

Red Hat OpenShift

- Enterprise product from Red Hat
- Comes with long-term support, security updates, and certified add-ons
- Uses Red Hat Enterprise Linux CoreOS
- Tailored for production workloads and mission-critical environments
  [kubernetes.dev]

Why Use OKD?

You might choose OKD if you want:

- A **free** but powerful Kubernetes distribution
- A platform that mirrors OpenShift but without licensing costs
- A learning environment for cloud-native technologies
- Access to cutting-edge Kubernetes advancements
  [bing.com], [bing.com]

Summary

OKD is essentially "OpenShift without the enterprise layer."
It provides the same Kubernetes foundation, many of the same developer tools, and a rich Operator ecosystem—but maintained by the community rather than backed by Red Hat support.

It's ideal for: Developers, Educators, Open-source enthusiasts, Organizations experimenting with Kubernetes-based platforms

Deploy a Node.js Backend to CSC Rahti

- Using the Web GUI (No CLI commands like oc (openshift client))

1. Log in to the Rahti Web Interface

1. Go to **https://rahti.csc.fi**
2. Choose the **Rahti web user interface** and log in with your CSC credentials. [github.com]

---

2. Create a Rahti Project

You must create a project before deploying anything.

1. In the Rahti web console, click **Create Project**.
2. Enter:
    - **Project Name** (becomes part of your app's URL)
    - **Display Name** (optional)
    - **Description** → must contain:
    - csc_project:<your CSC project number>
      NOTE: Rahti will not allow creation without linking it to a valid CSC computing project.

---

3. Switch to the Developer Perspective (at least old Web GUI had two perspectives)

1. On the left sidebar, select **Developer** (instead of Administrator).
2. Ensure your newly created project is selected in the project dropdown.

(This matches the workflow described in CSC's "Deploy from Git" tutorials.)
[docs.csc.fi – Deploy from Git]

---

4. Start Deployment Using "Import from Git"

CSC provides a simple, official flow for using a Git repo as the source for building and deploying apps:

- In the top-right corner, click **+ Add** → choose **Import from Git**.
- Paste your GitHub repo URL (HTTPS).
- Rahti will clone and analyze the repository:
    - It detects if there is a **Dockerfile** at the root.
    - If your Dockerfile is **not at the root**, open **Advanced Git Options** and set:

    - **Context Directory** → folder containing the Dockerfile
      (This is supported in the GUI according to the Deploy-from-Git guide).

- Under **Advanced Git Options**, you may also:

  - Select branch/tag/commit
  - Add authentication if the repo is private

1. After the validation succeeds, click **Create**.

This triggers Rahti to set up:

- A **BuildConfig** (pulls from your GitHub repo)
- An **ImageStream** (stores the built Docker image)
- A **Deployment** (runs your Node backend)
- A **Service** (exposes it inside the cluster)
- A **Route** (exposes it publicly via HTTPS)

CSC confirms that OpenShift automatically creates these objects when deploying from Git via the GUI.
[csc-guide-…ahtiapp.fi – Create static Webserver in Rahti]

---

5. Rahti Builds Your Docker Image

After clicking **Create**, Rahti will:

- Launch a build based on your **Dockerfile**
- Build the container image in its internal registry
- Deploy it automatically once the build succeeds

You can monitor the build using:

- Left navigation → **Builds**
- Click your build → **Logs**

---

6. Verify the Deployment in the Topology View

1. Click **Topology** in the Developer menu.
2. You will see a circular icon representing your Node backend application.
    - Click it to see:
        - Pod status

- Build history
- Environment variables
- Resource usage

---

7. Find the Public URL (Route)

Every publicly exposed service gets a **Route**.

1. In the node details panel (right side), look under **Routes**.
2. You'll see a link such as:

https://<app-name>-<project>.rahtiapp.fi

CSC documentation confirms that the Route is visible in the GUI under the project resources and provides the external URL.

Click it to access your running Node.js backend.

---

8. (Optional) Enable Automatic Redeployments via GitHub Webhooks

If you want Rahti to rebuild every time you push to GitHub:

1. In Rahti GUI → go to **Builds → your build → Configuration**
2. Copy the **GitHub webhook URL** provided
   (CSC shows exactly where to copy this in the web interface.)
   https://docs.csc.fi/cloud/rahti/tutorials/webhooks/
3. In GitHub → Repository → **Settings → Webhooks → Add Webhook**
4. Paste the webhook URL
5. Choose **Content-Type: application/json**

Rahti will now automatically rebuild and redeploy your backend on every push.

✅ Summary of the Entire GUI-Only Process

Step	Action	Source
1	Log in to Rahti web UI	[github.com]
2	Create Project with CSC project number	[github.com]
3	Switch to Developer mode (at least in older GUI	[docs.csc.fi]
4	Click *Import from Git* and give GitHub URL	[docs.csc.fi]
5	Configure branch/context under Advanced Git Options	[docs.csc.fi]
6	Click *Create* → Rahti builds Dockerfile automatically	[docs.csc.fi]
7	Check Application → Topology	[csc-guide-...ahtiapp.fi]
8	Open public Route URL	[csc-guide-...ahtiapp.fi]
9	(Optional) Set GitHub Webhooks via GUI	[csc-guide-...ahtiapp.fi]

Node.js backend Dockerfile sample for CSC Rahti

```
# ---------- Builder stage ----------
FROM node:20-alpine AS builder

# Create app directory
WORKDIR /opt/app

# Install dependencies first (leverages Docker layer cache)
# Copy only manifest files to avoid invalidating the cache unnecessarily
COPY package*.json ./

# Install clean, production-locked dependencies
# If you have dev dependencies for build steps (e.g., TypeScript), use `npm ci`
RUN npm ci

# Copy the rest of the source
COPY . .

# If you transpile (e.g., TypeScript) or bundle, do it here:
RUN npm run build

# ---------- Runtime stage ----------
FROM node:20-alpine

# Create a non-root friendly workspace.
# OpenShift/OKD (Rahti) runs containers with a random UID; ensure group 0 can
write.
WORKDIR /opt/app
COPY --from=builder /opt/app /opt/app

# Ensure permissions are compatible with random UID:
# - give group 0 ownership and write permission
# - make group permissions match user permissions recursively
RUN chgrp -R 0 /opt/app \
&& chmod -R g=u /opt/app

# Environment
ENV NODE_ENV=production \
    PORT=8080

# If you produce a build artifact (e.g., ./dist), you can prune dev deps:
# RUN npm ci --omit=dev
# Or if using the builder result, keep only what you need:
# (In that case, adjust the copy to only bring built files and prod
node_modules.)

# App listens on 8080 inside the container
EXPOSE 8080

# HEALTHCHECK is optional but recommended (adjust path to your health endpoint)
# HEALTHCHECK --interval=30s --timeout=3s --start-period=30s \
#   CMD wget -qO- http://127.0.0.1:8080/health || exit 1

# IMPORTANT: Do NOT set a fixed USER here; Rahti injects a random UID.
# Just ensure filesystem perms allow any UID in group 0 to write where needed.

# Start the app (adjust if you use a different start command)
CMD ["npm", "start"]
```

Rahti vs. cPouta — Side-by-Side Comparison

1. Core Purpose (Rahti and cPouta are maybe most used CSC offered options on our courses)

Rahti

- A **container orchestration service** for running applications in containers. (**PaaS**).
- Built on **OKD**, the community Kubernetes distribution behind OpenShift. [youtube.com]
- Focuses on **easy deployment of web applications, APIs, scientific tools, and containerized workflows**. [youtube.com]

cPouta

- An **Infrastructure-as-a-Service (IaaS)** cloud.
- Provides **virtual machines, networks, storage**, and the freedom to build full custom environments.
- Based on **OpenStack**, not Kubernetes. [research.csc.fi] = no docker (by default, unless YOU install it and run it on you virtual machine).

---

2. Level of Control

Rahti (Higher-level)

- You manage **applications**, not servers.
- No container root access; runs workloads as **non-privileged users** for security. [intowindows.com], [youtube.com]
- Infrastructure is abstracted away; scaling, updates, and routing are built in.

cPouta (Lower-level)

- You manage **virtual machines**, operating systems, patches, firewalls, etc.
- Full control over OS and configuration; this includes all **responsibilities for security, updates, access control, backups**. [research.csc.fi]

---

3. Typical Use Cases

Rahti is ideal for:

- Running **web applications or APIs** with built-in HTTPS, DNS, and routing. [intowindows.com], [easytweaks.com]
- Deploying **scientific software stacks** packaged as containers.
- Scalable, fault-tolerant apps using rolling updates and auto-scaling. [easytweaks.com]
- Situations where you want "**just run my app**" without server maintenance.

cPouta is ideal for:

- Running **custom compute workflows**, e.g. HPC, GPU jobs, I/O-heavy workloads. [research.csc.fi], [neic.no]
- Hosting full server environments, databases, or legacy apps requiring OS-level control.
- Building your own services (databases, backends, pipelines) from scratch.
- Infrastructure automation using Terraform/OpenTofu. [research.csc.fi]

---

4. Networking & Exposure

Rahti

- Provides **automatic DNS** such as *.rahtiapp.fi and **HTTPS certificates included**. [intowindows.com], [easytweaks.com]
- Uses Kubernetes **Routes** and built-in load balancing.

cPouta

- You manage all networking:
    - Floating IPs
    - Security groups
    - Firewall rules
- VMs are directly accessible from the public internet if assigned IPs. [research.csc.fi], [neic.no]
- Security hardening is 100% the user's responsibility.

---

5. Data Storage & Backup

Rahti

- Content stored in Finland but **not backed up**; user responsible for backups. [intowindows.com]
- Built for containerized workloads with persistent volumes available.

cPouta

- Offers volumes (block storage) attached to VMs.
- User is responsible for all backups and data protection. [research.csc.fi]

---

6. Security Model

Rahti

- **Multi-tenant application platform**; containers cannot run as root.

- Suitable for general-purpose apps but not ideal for sensitive data unless externally protected. [youtube.com]

cPouta

- ISO/IEC 27001:2022–certified environment. [research.csc.fi]
- Can process personal data depending on user's compliance.
- For highly sensitive data, CSC recommends **ePouta** instead. [research.csc.fi]

---

7. Ease of Use

Rahti

- Easier for users unfamiliar with system administration.
- Smooth UI for deployments and logs, automatic scaling, and built-in app lifecycle management. [easytweaks.com]

cPouta

- Requires Linux administration knowledge.
- Users manage OS, SSH access, updates, firewalling, etc. [research.csc.fi]
- More flexible but also more work.

Summary Table

Feature	Rahti	cPouta
Type	Container cloud (Kubernetes/OKD) [youtube.com]	IaaS virtual machines (OpenStack) [research.csc.fi]
Management Level	Applications	Infrastructure (VMs, networks, storage)
Security	No root containers, shared cluster [intowindows.com]	Full responsibility for VM security [research.csc.fi]
Best For	Web apps, APIs, scientific container workloads	HPC/GPU jobs, custom servers, legacy apps
Networking	Auto DNS + HTTPS built-in [intowindows.com]	Floating IPs + security groups managed manually [research.csc.fi]
Backups	User-managed (no automatic backup) [intowindows.com]	User-managed backups [research.csc.fi]
Skills Needed	Basic container familiarity	Linux & system administration
Flexibility	Medium (container constraints)	Very high (full VM control)

Which Should You Choose?

Choose Rahti if:

- You want to deploy containerized apps quickly.
- You prefer managed scaling, routing, and certificates.
- You don't need root access or OS control.

Choose cPouta if:

- You need full control of servers and networks.
- You run HPC, GPU, or complex system-level workloads.
- You need to build customized environments from scratch.

Haaga-Helia CSC instructions (in Finnish):

https://haagahelia.github.io/hh-csc-docs/ (Apart from text in images, translation semi-works!)