



James Carppe

# YachtOps: Become a Captain for Kubernetes in 120 Minutes

Get a production-ready Kubernetes cluster up and running with Portainer and Omni in under two hours.



Introduction



Provision a management environment



Configure external authentication



Set up Omni



Create your Talos machines





Create your Kubernetes cluster



Secure your cluster



 Upgrade the Kubernetes version Role-based Access Control (RBAC) Deploy an application Summary



# Introduction



James Carppe

---

## Welcome

Welcome to the YachtOps workshop! In this session we will be provisioning and configuring a Kubernetes cluster using Omni, managed by Portainer. By the end of the workshop you will have:



A management server running Docker Standalone with Portainer Business Edition.



A secured, production-ready three-node Kubernetes cluster running Talos Linux, provisioned via Omni and managed by your Portainer management server.



External authentication to your Portainer server configured via OAuth.



Role-based Access Control (RBAC) set up for your cluster.



A sample application deployed on your cluster.

## What you'll need



For this workshop we'll assume you have the following:

☐

A laptop or other computer that you can use to complete the lesson.

☐

An internet connection.

☐

A GitHub account. You'll use this as your OAuth provider. If you have another OAuth provider you prefer you can use that, but this workshop will assume you are using GitHub. You'll also use GitHub to deploy the sample application.

As part of the workshop you'll also need the following:

☐

A Digital Ocean account. This is for the management server and cluster nodes you'll create. We'll provide a referral link so you can get started with some credit. If you have another hosting provider you would prefer you can use them, but this workshop will assume you are using Digital Ocean (and has some steps and configurations that are Digital Ocean specific).

☐

An Omni account with Sidero Labs. The workshop will walk you through the setup process for this.

☐

A Portainer Business Edition license. If you do not already have a license, you can [sign up here](#).

Let's get started!



# Provision a management environment



James Carppe

---

The first thing we need when setting up a cluster with Portainer is, well, Portainer. In this lesson you will:

1

Provision a server to act as a "management server" that will stand outside of your Kubernetes cluster.

2

Install Docker Standalone on the server.

3

Install Portainer Business Edition and complete the initial setup.

**START**

---

1

## Provisioning a server on Digital Ocean

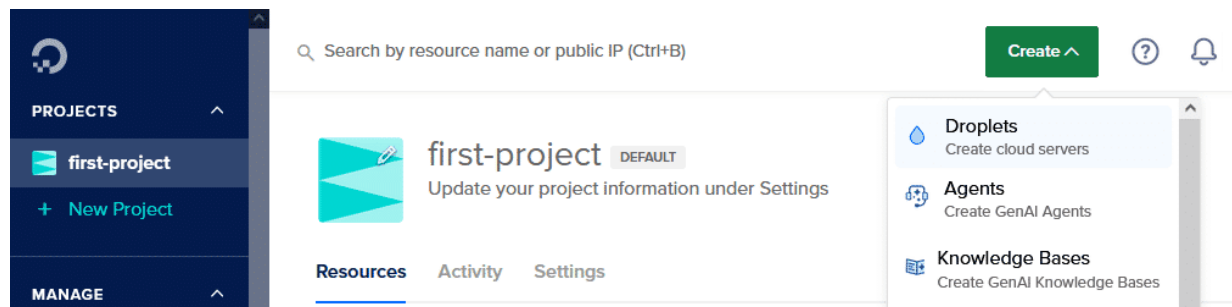


For this workshop, we're going to spin up a Ubuntu Server VM in Digital Ocean to act as our "management server". We'll install Docker on this server, then Portainer itself.



**If you are using a different provider or have already created your management server, you can skip ahead to the next section.**










Log into your Digital Ocean account. Click the **Create** button, then select **Droplets**.



Choose the **region** you want to deploy in and optionally the **datacenter** within that region. Make a note of what you choose as we'll want to use the same region and datacenter when we create the Kubernetes cluster. Also make note of the **VPC Network** - we'll need that too.




## Choose Region

 New York	 San Francisco	 Amsterdam
 Singapore	 London	 Frankfurt
 Toronto	 Bangalore	 Sydney

## Datacenter

London • Datacenter 1 • LON1

 **Tip: Select the datacenter closest to you or your users** [Dismiss](#)

Avoid any potential latency by selecting a region closest to you - a region is a geographic area where we have one or more datacenters.

VPC Network - default-lon1







DEFAULT

All resources created in this datacenter will be members of the same VPC network. They can communicate securely over their Private IP addresses.

Next we choose the image. For this workshop we'll pick Ubuntu 24.04 (LTS) x64.

## Choose an image

OS Marketplace (281) Custom images

 Ubuntu	 Fedora	 Debian	 CentOS	 AlmaLinux	 Rocky Linux
---	---	---	---	--	---

Version

24.04 (LTS) x64 



Now we pick the size of the VM. Since this VM is only going to run the Portainer server and not any actual workloads, we can choose a Basic type, Regular CPU type and the 2GB / 2 CPUs plan.

## Choose Size

Need help picking a plan? [Help me choose](#)

### Droplet Type

SHARED CPU	DEDICATED CPU			
<b>Basic</b> (Plan selected)	General Purpose	CPU-Optimized	Memory-Optimized	Storage-Optimized

Basic virtual machines with a mix of memory and compute resources. Best for small projects that can handle variable levels of CPU performance, like blogs, web apps and dev/test environments.

### CPU options

☒ **Regular**  
Disk type: SSD

☐ **Premium Intel**  
Disk: NVMe SSD

☐ **Premium AMD**  
Disk: NVMe SSD

<b>\$4/mo</b> \$0.006/hour	<b>\$6/mo</b> \$0.009/hour	<b>\$12/mo</b> \$0.018/hour	<b>\$18/mo</b> \$0.027/hour	<b>\$24/mo</b> \$0.036/hour	<b>\$48/mo</b> \$0.071/hour
512 MB / 1 CPU 10 GB SSD Disk 500 GB transfer	1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer	2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer	2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer	4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer	8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer

[Show all plans](#)

Scroll down to the **Choose Authentication Method** section. Here you can decide how you'll access this server for the initial setup. If you have a SSH key pair you can use, we recommend selecting this option and providing your key pair. Otherwise you can choose to use a password instead.





**You will need to have SSH key authentication configured to create the Kubernetes cluster machines later on in this course, so we recommend configuring this now.**

### Choose Authentication Method ?



#### SSH Key

Connect to your Droplet with an SSH key pair



#### Password

Connect to your Droplet as the "root" user via password

#### Add a public SSH key

SSH keys are a more secure method of logging into an SSH server, because they are not vulnerable to common brute-force password hacking attacks.



**We can walk you through setting up your first SSH key**

[Add SSH Key](#)

Finally, scroll down to the **Finalize Details** section and give your VM a **Hostname**, for example manager.



## Finalize Details

### Quantity

Deploy multiple Droplets with the same configuration.

—

1 Droplet

+

### Hostname


Give your Droplets an identifying name you will remember them by.

manager

### Tags

Type tags here

### Project

 first-project

When you're ready, click **Create Droplet**. Once the creation completes you will be returned to the list of droplets, where you'll see yours provisioning. When it completes, you'll see the server's IP address listed.

**Resources** Activity Settings

DROPLETS (1)

  manager

68.183.41.75



Upsize



CONTINUE



## Installing Docker Standalone

Now that we have our management server provisioned, let's set it up with Docker Standalone.

Use your favorite SSH application to log into the server with your root credentials. You should see something like the following:



```
james@SPRINGHAWK:~$ ssh root@68.183.41.75
The authenticity of host '68.183.41.75 (68.183.41.75)' can't be established.
ED25519 key fingerprint is SHA256:8hhiedfp0vrCVMX6c0kPNDkvygzUriwF7HxrrcaQcBc.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '68.183.41.75' (ED25519) to the list of known hosts.
root@68.183.41.75's password:
Welcome to Ubuntu 24.04.1 LTS (GNU/Linux 6.8.0-51-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Wed Mar 19 04:20:50 UTC 2025

System load:  0.0                Processes:            122
Usage of /:   3.1% of 57.08GB    Users logged in:     0
Memory usage: 9%                IPv4 address for eth0: 68.183.41.75
Swap usage:   0%                IPv4 address for eth0: 10.16.0.5

Expanded Security Maintenance for Applications is not enabled.

162 updates can be applied immediately.
52 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@manager:~# |
```

From here, the first thing we should do is ensure the server's packages are up to date. As this is Ubuntu, we can do this with apt. Run the following two commands:



```
apt update
apt upgrade -y
```

The first command updates the apt repository with the latest list of packages. The second command upgrades your currently installed packages to their latest versions according to the repository we just updated. This may take a few minutes to complete, and we generally recommend rebooting the server once it completes in order to ensure all updated packages are loaded.

```
reboot
```

Once the server completes rebooting, log back in with SSH. We're now ready to install Docker Standalone. For this we'll rely on [Docker's official installation instructions](#) for Docker Engine on Ubuntu, the commands for which are summarized below:

```
apt install ca-certificates curl
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable \
  tee /etc/apt/sources.list.d/docker.list > /dev/null
```



```
apt update
apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docl
```

Alternatively you can run this helper script to perform the necessary steps:

```
curl -sSfL https://get.docker.io | sh
```

Once this is complete, Docker should be installed. You can check this by running:

```
docker ps
```

You should see something similar to the below:

```
root@manager:~# docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED   STATUS    PORTS     NAMES
```



If you do, then Docker is up and running.

CONTINUE

3

## Installing Portainer

We have a server, we have Docker installed, so now we need to install Portainer itself.

Installation instructions for Portainer can be found [in our documentation](#). Since we're installing on Docker Standalone, we just need to run these two commands:

```
docker volume create portainer_data

docker run -d -p 8000:8000 -p 9443:9443 --name portainer --restart=always
```

The first command creates a persistent data volume for Portainer's configuration. The second command starts the Portainer container.

Once the second command completes you can check Portainer's status by running `docker ps` again. You should see the Portainer container listed:



```
root@manager:~# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED
b881e36f9355	portainer/portainer-ee:1ts	"/portainer"	28 seconds ago

Now that the container is running, we need to connect to the Portainer UI in order to complete the setup.



**This must be done within 5 minutes of starting the Portainer container for the first time. This is a security measure. If the initial setup has not been completed within 5 minutes of starting, the container will stop listening and will need to be recreated.**


Open your web browser of choice and navigate to the following URL (replace `ipaddress` with the IP address of your management server):

```
https://ipaddress:9443/
```

Accept the certificate warning (Portainer generates a self-signed certificate during installation) and continue, and you'll be presented with the initial setup.



First we must set up an initial administrator user. You can choose to use a different username or stick with the default of admin. Enter a secure password, confirm it, and click **Create user**.



▼ **New Portainer installation**

Please create the initial administrator user.

Username

Password

Confirm password  ✓


⚠ The password must be at least 12 characters long. ✓

☒ Allow collection of anonymous statistics. You can find more information about this in our [privacy policy](#).

> **Restore Portainer from backup**


Next we'll enter the license key. Paste your key into the box provided and click **Submit**.





### License registration

Please register your Portainer license.


 Your license key should start with "2-" or "3-".

License

Submit

[Don't have a license?](#)

Once this is complete, you'll be logged into Portainer. The local Docker environment will be automatically detected and configured, and you're ready to proceed.



portainer.io  
BUSINESS EDITION

Home


Environment: / None selected

Administration

- User-related
- Environment-related
- Registries
- Licenses
- Logs
- Notifications
- Settings

Environment Wizard


Quick Setup

 Environment Wizard

#### Welcome to Portainer


We have connected your local environment of Docker to Portainer.

Get started below with your local portainer or connect more container environments.



#### Get Started

Proceed using the local environment which Portainer is running in



#### Add Environments

Connect to other environments



## Summary

In this lesson we have:

- ☐ Provisioned a management server.
- ☐ Installed Docker on the management server.
- ☐ Installed Portainer and completed the initial setup.

In the next lesson we'll set up external authentication for Portainer using OAuth.



# Configure external authentication



James Carppe

---

We're up and running with a Portainer management server, but right now we only have one user - the internal administrator. In the real world, you'll want to have multiple users with different levels of access using your cluster. While this can be done with Portainer's internal authentication, when you have an existing authentication provider it makes more sense to leverage that - giving you one place where you manage your users.

In this lesson we will:

1

Configure GitHub as an OAuth authentication provider.

2

Set up Portainer to use GitHub as the authentication method.

3

Test out access to Portainer as a GitHub user.

**START**

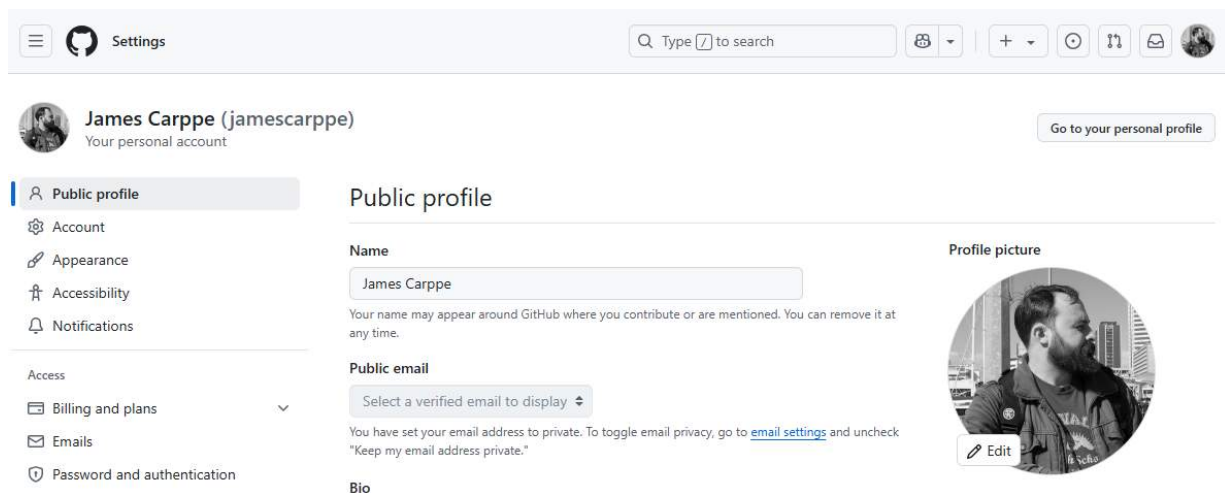
---



## Setting up GitHub as an OAuth provider

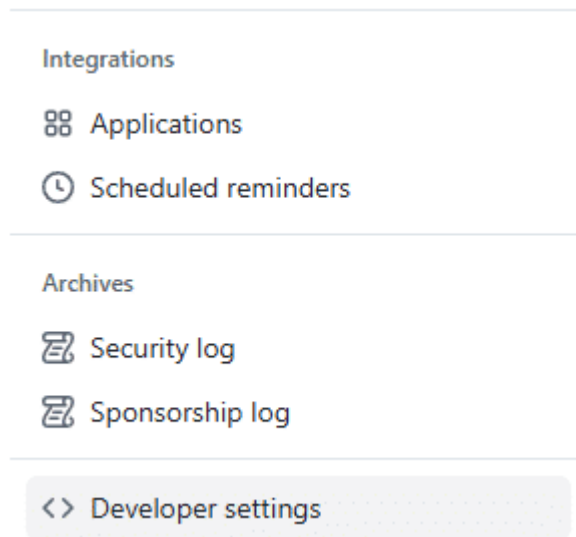
Configuring GitHub to provide OAuth authentication is a relatively straightforward process. You don't need a commercial account - a free GitHub user is plenty to get started. So let's do that.

Log into GitHub as your user, then click your profile picture in the top right and select **Settings**. You'll be taken to your profile page.

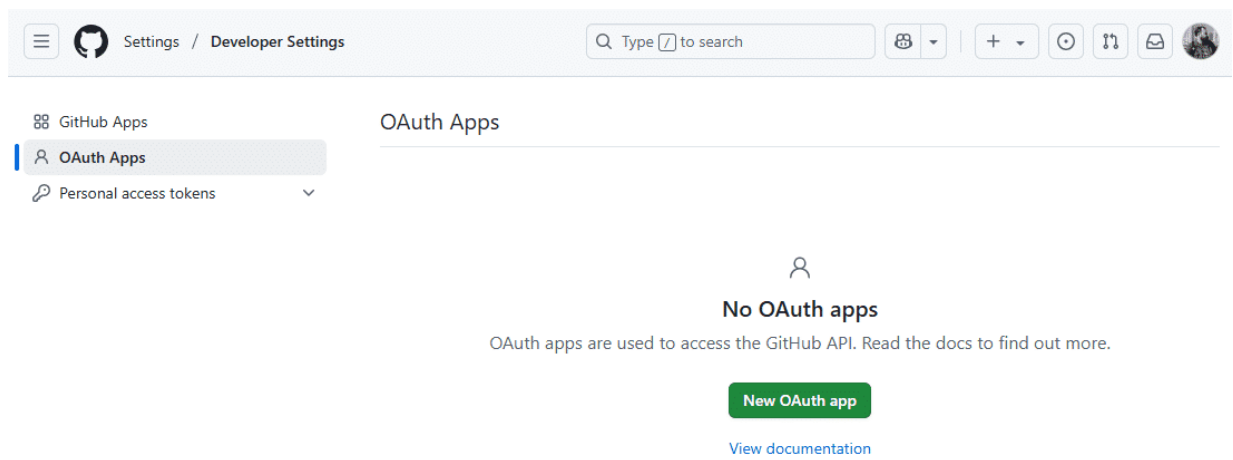


From here, scroll down and click **Developer settings** at the bottom of the left menu.





Next, select **OAuth Apps** from the left menu. This will take you to a list of your OAuth apps configured in GitHub. In my case, I have none set up currently.



We'll create a new app now for Portainer. Click the **New OAuth app** button then fill out the form.

**Application name** should identify the application you're using OAuth with - for example, Portainer. The **Homepage URL** and the **Authorization callback URL** should be the URL



that you use to access the Portainer UI, including the `https://` prefix and the port. The **Application description** is optional.

## Register a new OAuth app

---

**Application name \***

Portainer

Something users will recognize and trust.

**Homepage URL \***

`https://68.183.41.75:9443/`

The full URL to your application homepage.

**Application description**

OAuth for Portainer

This is displayed to all users of your application.

**Authorization callback URL \***

`https://68.183.41.75:9443/`

Your application's callback URL. Read our [OAuth documentation](#) for more information.

☐ **Enable Device Flow**

Allow this OAuth App to authorize users via the Device Flow.

Read the [Device Flow documentation](#) for more information.

---

**Register application**

[Cancel](#)

Once you have the form filled out as required, click **Register application**. The application will be created and you'll be taken to the details page. From here, take note of the **Client ID** value displayed - we'll need this for the next step.




General

Optional features

Advanced

## Portainer

 **jamescarppe** owns this application.

Transfer ownership

You can list your application in the [GitHub Marketplace](#) so that other users can discover it.

List this application in the Marketplace

0 users

Revoke all user tokens

Client ID

0v231ieq0JBG7xQFNOxs

Client secrets

Generate a new client secret

You need a client secret to authenticate as the application to the API.


To go along with the Client ID, we need a Client secret. Click the **Generate a new client secret** button. If you have two factor authentication set up on your GitHub account, you may be asked to reauthenticate at this point.

Once you have done so, a **Client secret** will be created. Take careful note of this value, as you won't be able to display it again after this.


## Client secrets

Generate a new client secret

Make sure to copy your new client secret now. You won't be able to see it again.



Client secret

✓ 348b3bffeb3fa81b0508de46df566907d98586c5 

Added now by jamescarppe

Never used

You cannot delete the only client secret. Generate a new client secret first.

Delete



This is all we need from GitHub. You can now switch back to Portainer to continue the OAuth setup there.

CONTINUE

2

## Configuring Portainer for OAuth

With an OAuth application created in GitHub and our client ID and secret obtained, we're ready to configure Portainer for OAuth.

In Portainer, scroll down and expand the **Settings** menu at the bottom left and select **Authentication**.



The screenshot displays the Portainer.io Business Edition web interface. On the left is a dark sidebar with the Portainer logo and navigation links. The 'Home' link is highlighted. Below it, a box indicates 'Environment: / None selected'. The 'Administration' section is expanded, showing options like 'User-related', 'Environment-related', 'Registries', 'Licenses', 'Logs', 'Notifications', and 'Settings'. The 'Settings' option is selected, and the 'Authentication' sub-option is highlighted. The main content area on the right is titled 'Environments' and shows a 'Home' button with a refresh icon. Below this, there's a section for 'Environments' with a button to 'Click on an environment to manage'. Two dropdown menus are visible: 'Platform...' and 'Connecti...'. A card for the 'local' environment is shown, indicating it is 'Up' and has '0 stacks' and '1 container'. The 'local' environment is currently selected.

portainer.io  
BUSINESS EDITION

Home

Environment: / None selected

Administration

- User-related
- Environment-related
- Registries
- Licenses
- Logs
- Notifications
- Settings

General

Authentication

Shared Credentials

Edge Compute

Get Help

Environments

Home

Environments

Click on an environment to manage

Platform... Connecti...

local Up 202

Group: Unassigned

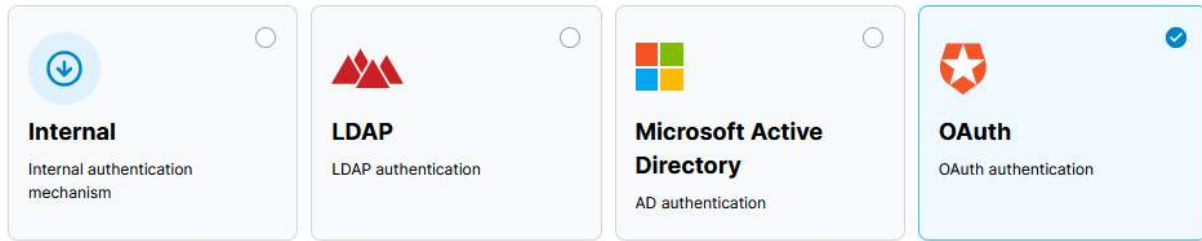
0 stacks 1 container

2 CPU 2.1 GB

Here you'll see the authentication options for your Portainer instance. The **Internal** method is currently selected, but we want to change this so select **OAuth** instead.



#### Authentication method



When OAuth is selected, the options below the selection will change. Here we want to ensure that **Use SSO** is enabled, **Hide internal authentication prompt** is disabled, and **Automatic user provisioning** is enabled.

#### Single Sign-On

Use SSO  ☒

Hide internal authentication prompt ☐

#### Automatic user provisioning

With automatic user provisioning enabled, Portainer will create user(s) automatically with the standard user role. If disabled, users must be created beforehand in Portainer in order to login.

Automatic user provisioning ☒

Since we're using GitHub as our OAuth provider, we can select that option now, and fill in the **Client ID** and **Client secret** fields with the appropriate values from our previous step.



[illegible]

CONTINUE

Once you're logged out, return to the URL for your Portainer instance. The login screen should look a little bit different now.





## Log in to your account

Welcome back! Please enter your details

 Login with GitHub

or

Use internal authentication

Click **Login with GitHub** to log back in, and enter your GitHub credentials (and two-factor code) if asked.





Sign in to **GitHub**  
to continue to **Portainer**

Username or email address

Password

[Forgot password?](#)

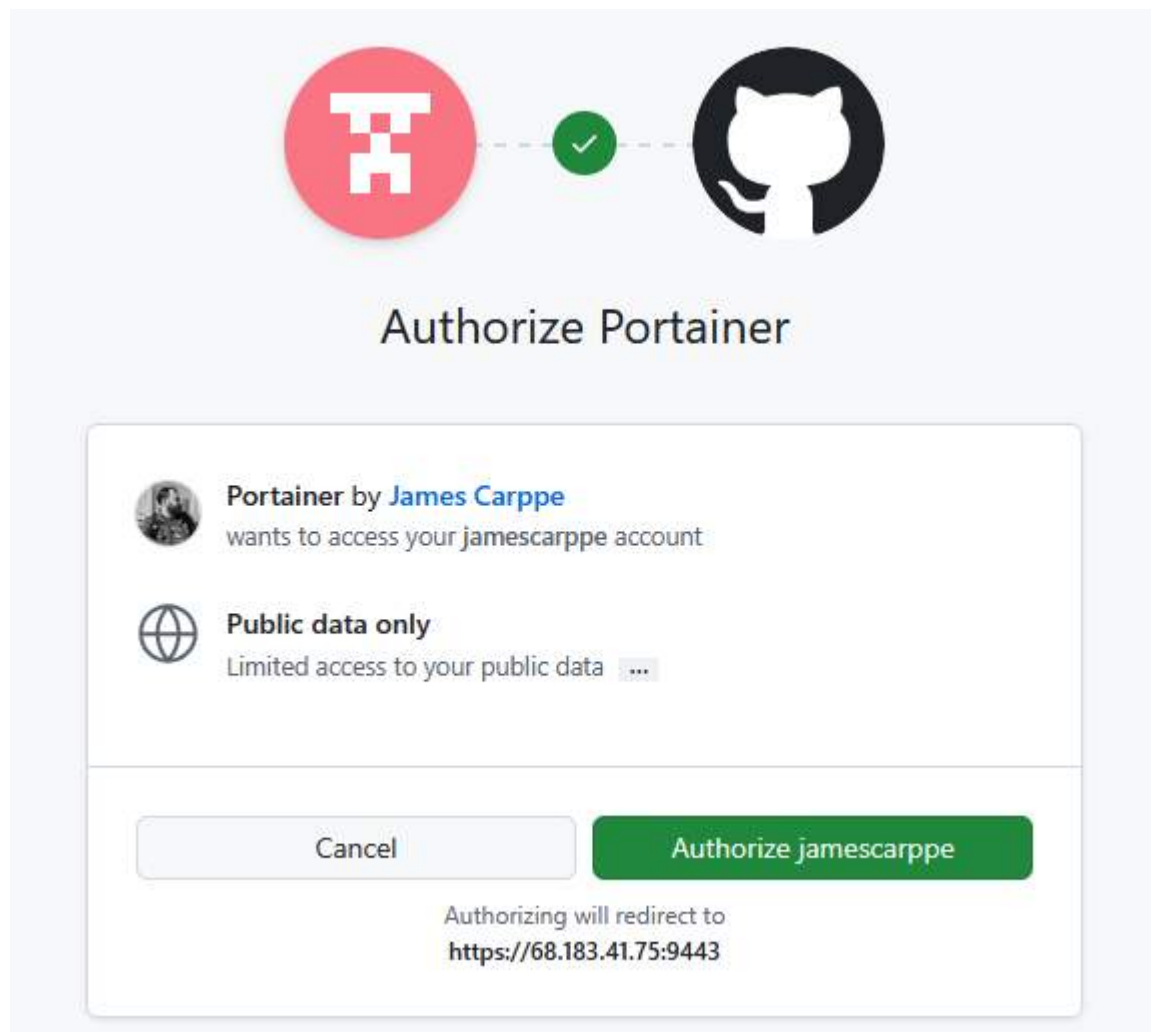
Sign in

[Sign in with a passkey](#)

New to GitHub? [Create an account](#)

As this is the first time you're logging in to this OAuth application, you will be asked to authorize access to your GitHub account from your OAuth application. Click the **Authorize** button to proceed.

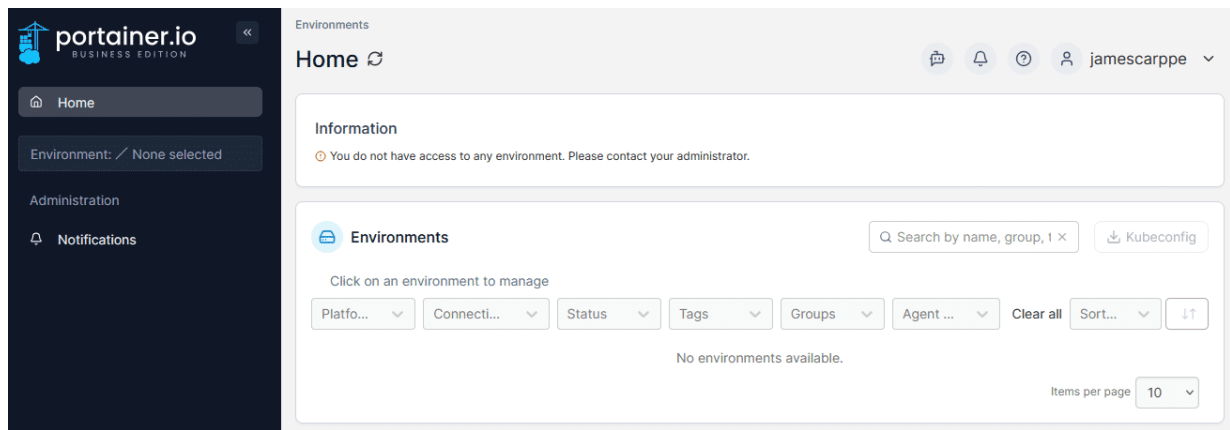




With authorization granted, the login will proceed and you will be taken to the Portainer dashboard, as your GitHub user.

What you may note is that you don't have access to very much at all as this user.





By default, new users logging into Portainer for the first time have no access to any environments or other settings. Once we have our Kubernetes environment configured, we'll configure access to it for this user.

For now, we've shown that OAuth is working and you can log into Portainer as expected. So let's log out again and back in as the local administrator account you created during the initial Portainer setup. You can do this by selecting **Use internal authentication** at the login page instead of Login with GitHub.

CONTINUE

4

## Summary

In this lesson we have:





Created an OAuth application in GitHub.



Configured our Portainer server to use the GitHub OAuth application as the authentication provider.



Tested logging into Portainer with our GitHub user.

In the next lesson we'll set up an Omni account and configure Portainer to interact with it.



# Set up Omni



James Carppe

---

To provision our Kubernetes cluster we'll be using Portainer's integration with Sidero's Omni platform. To do this, we'll need to set up an Omni account.

In this lesson we will:

- 1 Sign up for an Omni account with Sidero.
- 2 Create a Service Account within our Omni account.
- 3 Download the Digital Ocean machine image we will use to provision our cluster machines.
- 4 Add the Omni Service Account to our Portainer installation.

**START**

---



## Creating an Omni account

Sidero offer a free trial of Omni SaaS that you can sign up for [on their website](#). Head there and fill out the form, then click **Start Trial**.

### Sign up for a free Omni trial - no credit card needed.

First Name

Last Name

Email

What's your role?

Team member

Omni Domain

https://

.omni.siderolabs.io

Omni plan

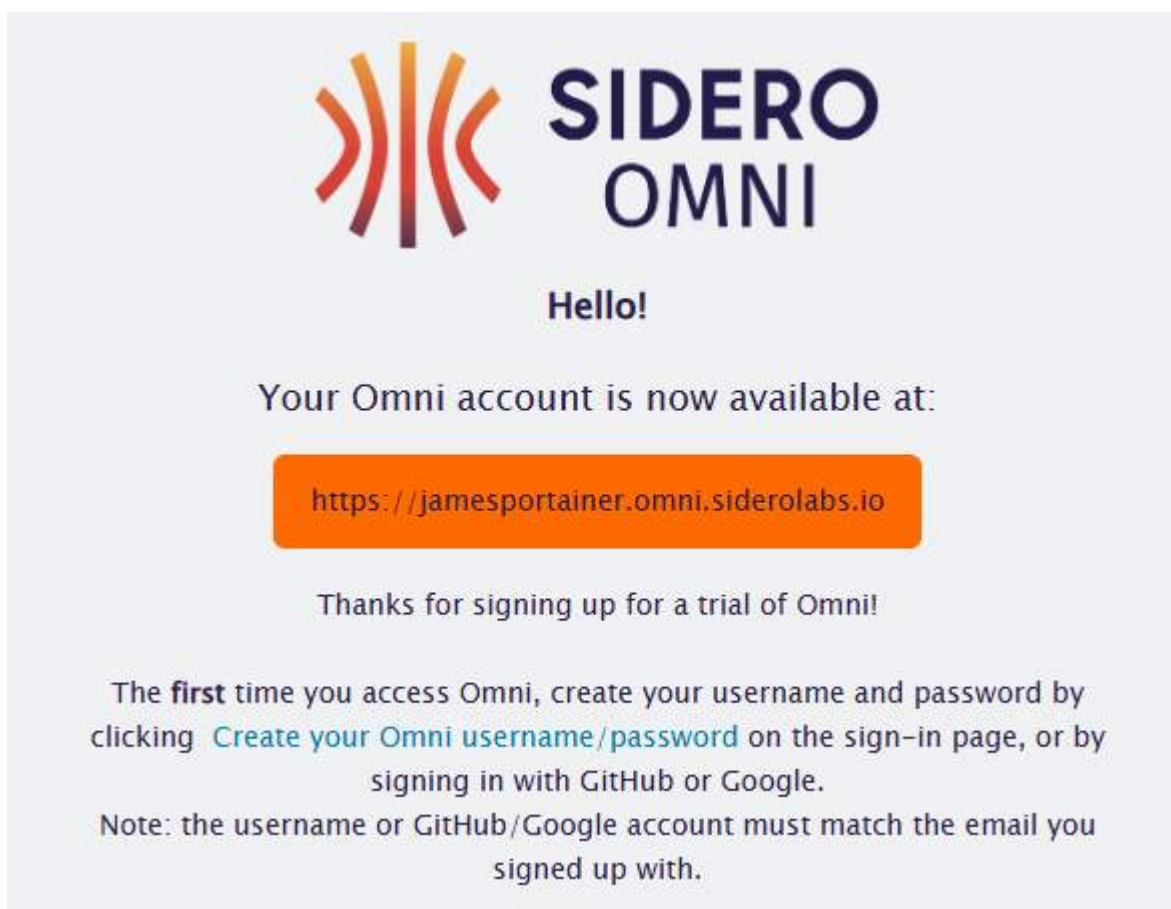
Business

☐ By clicking here, I state that I have read and agree to the [terms and conditions](#).

Start Trial



Once you've completed the signup you'll receive an email that will include your Omni account address.



Go to the URL provided, then click on **Create your omni username/password**.





## Welcome

Log in to continue to Sidero Labs Omni.



[Forgot password?](#)

Continue

First time logging in? [Create your Omni username/password](#)

On the Register form, fill in the email address you used when signing up and set a strong password, then click **Continue**.





## Register Account

Sign Up to continue to Sidero Labs Omni. If registering the first account for this Omni instance, the email **\*MUST\*** match the email you used to sign up for Omni.



Continue

Already have an account? [Log in](#)

You should now receive a second email to verify your email address. Click on the link provided in the email to do so.





## Welcome to Sidero Labs Omni!

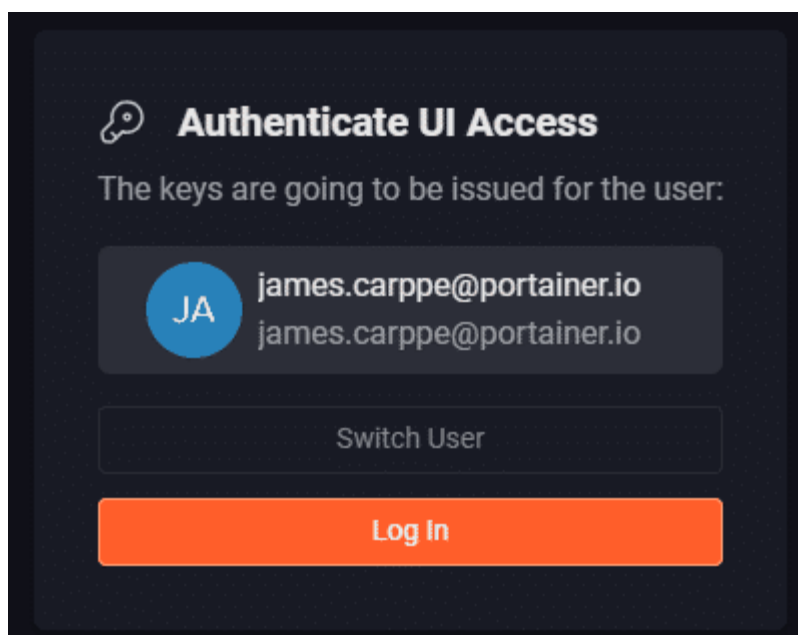
Thank you for signing up. Please verify your email address by clicking the following link:

[Confirm my account](#)

If you are having any issues with your account, please don't hesitate to contact us by replying to this mail.

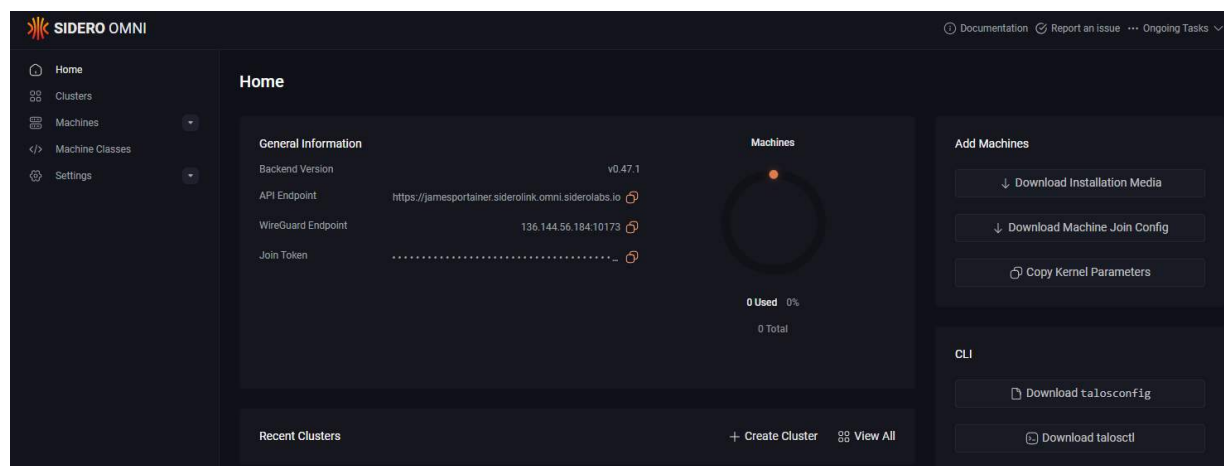
Thanks!  
**Sidero Labs Omni**

Once verified you will be able to return to the Omni login page and authenticate UI access. Click **Log in** to complete the login process.





You should now arrive at the Omni dashboard.



CONTINUE

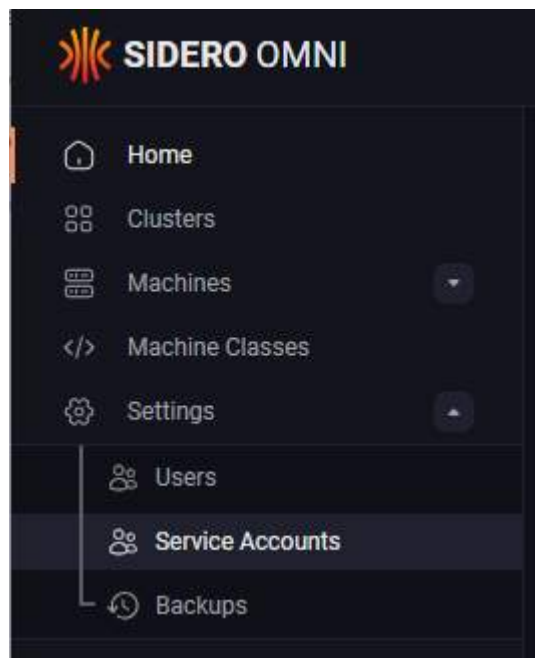
2

## Creating an Omni Service Account

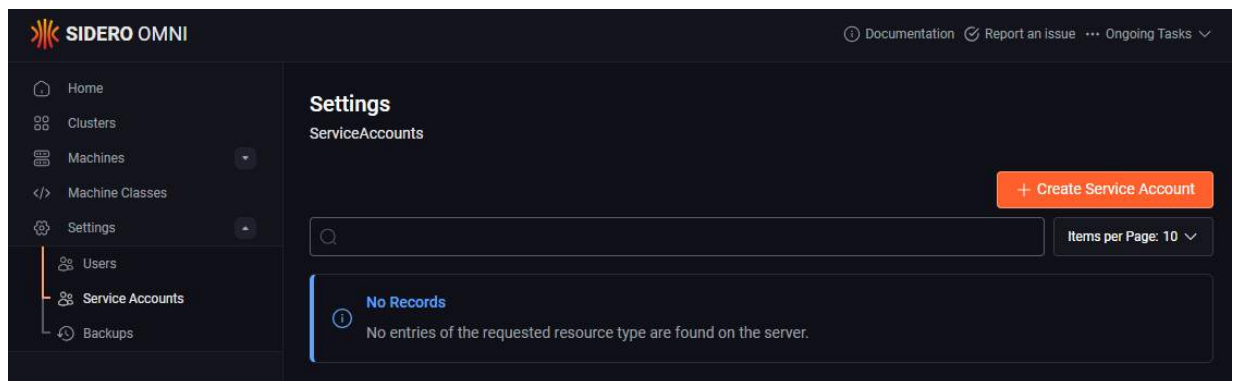
Now that our Omni account is set up, we need to configure a Service Account. This Service Account is how Portainer communicates with Omni in order to provision and manage clusters.

From the Omni dashboard, expand the **Settings** menu on the left and click **Service Accounts**.



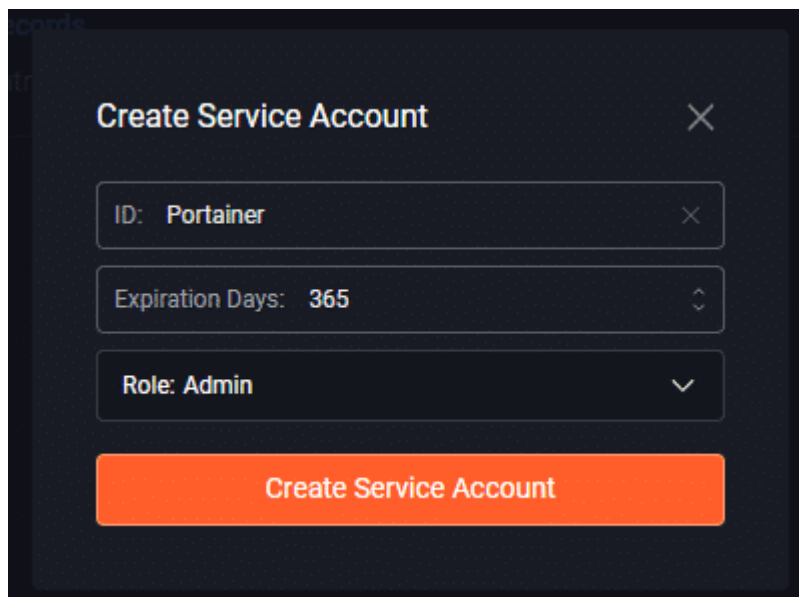


On the **Service Accounts** page we'll see that we have none currently created. So let's create one. Click the **Create Service Account** button.



Complete the form that appears. For **ID** you can use a name to identify the application that is going to be using the Service Account (in this case, Portainer). Configure the **Expiration Days** to a value that makes sense to you, and ensure the **Role** is set to Admin. When you're ready, click **Create Service Account**.



A dark-themed modal dialog box titled "Create Service Account" with a close button (X) in the top right corner. It contains three input fields: "ID: Portainer" with a clear button (X), "Expiration Days: 365" with a dropdown arrow, and "Role: Admin" with a dropdown arrow. At the bottom is a large orange button labeled "Create Service Account".

records

tr

### Create Service Account

ID: Portainer

Expiration Days: 365

Role: Admin

Create Service Account

The Service Account will create and you will be provided with a block of code containing a pair of environment variables - OMNI\_ENDPOINT and OMNI\_SERVICE\_ACCOUNT\_KEY. Copy both now - they will not be shown again and are needed for the next step.



Create Service Account

✔ Service Account Was Created

Dismiss

Set the following environment variables to use the service account:

```
export OMNI_ENDPOINT=https://jamesportainer.omni.siderolabs.io
export OMNI_SERVICE_ACCOUNT_KEY=eyJ0YXV1IjoiaUg9ydGFpbmVyIiwicGdwX2t1eSI6Ii0tLS0tQkVHSU4gUEdQIFBSSVZBVEUgS0VZIEJMT0NLLS0tLS1cb1xueFZnRV
o5dFpFaF1KS3dZQk3BSGFSDzhCQVfkQTFzQ1hiZ01LdUdBWnladkVaS3U2TXVFV3hic
nVvOWVoXG5ZdC9nTHNjMVhEOEFBUURmZW5rMFUxUnNPSmNBOTBmQWFOzZBQytwdmI2
QzhyMG5NakVPRnR3NVF3MmxcbnpTbzhVRz15ZEdGcGJtVn1RSE5sY25acFkyVmhzMk5
2ZFclMEExtOXRibWt1YzJsa1pYSnZMbVJsZGo3Q1xua2dRUUZnb0FSQVdDwjl0WkVnV0
pBZUV6Z0FRTEENRY01DWkNOVS8xRERGa1ZUQU1WQ0FvRUZnSUJBQU1aXG5BUUt1QXJZ
UFSWWhCTGJ4NFuQyztVWRLdS91R28xVC9VTU1XU1ZNQUFCYUznRctJN1JSUwVUFZq
Mw1cbjYXU5Mw1vQXNGUQU4QUmrN0t4U3B1T11SVDN5dVFCQUpPSjN4MEVsUDIrQ1Q
rN2hFb1diY3FkQzFLV1xuTmFWMHdGQm1xQ1NDU3IwQXgxMEVaOXRaRWhJS0t3WUJCQU
dYV1FFRkFRRUhRRmVmSk5EOTZ6MwdhaGhYXG4zcno5d2R4WjdCYitpUjg3akpMMVdkb
VMzZmNGQXdFSUJ3QUEvM0pVb1A3V0QxYmJSTE9pUThvUVJzekpcbmNSRnpS2VmaH1P
ZkJoRUVDVURvRDRMQ2ZnUV1GZ29BTUFXQ1o5dFpFZ1dKQWVFemdBbVFqV1A5UXd4W1x
uR1V3Q213d1dJUVMY0GVFSnd2ckZIU3J2M2hXT1UvMURERmtWVEFBQVJSQUJBShNLe1
ZENHRMVHZ5NkJKXG5iR1Y0SkpSNkdNdT1WN1BqOXpCZ2VXS3hQd11RQVFEYwZ4am0rW
XlaSU04T3FsdXBiaHdCbXdjUXBPRmRcbjhWRldQTFhDeEJQY0NnPT1cbj1hT2FNXG4t
LS0tLUVORCBQR1AgUFJJVkcFURSBURVkgQkxPQ0stLS0tLSJ9
```

Store the key securely as it will not be displayed again.

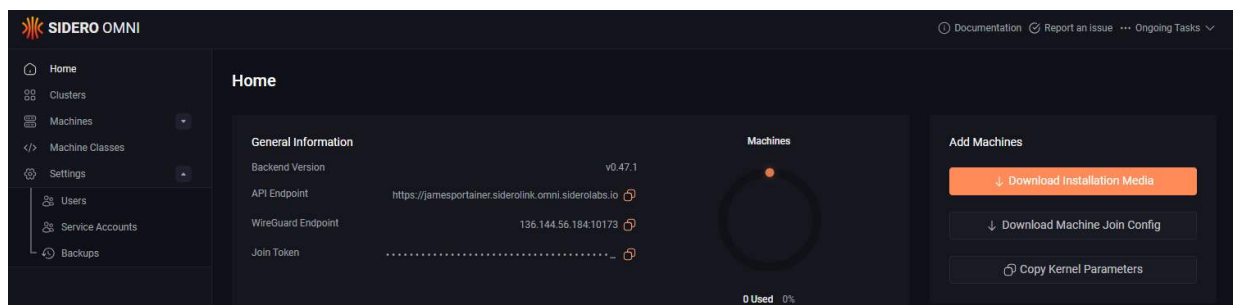
We're now ready to add the Service Account to Portainer. But before we do that, while we're in the Omni dashboard we can download the machine image we'll use for provisioning our cluster nodes.

CONTINUE



## Downloading the machine image

Return to the **Home** page of the Omni dashboard if you aren't there already. From here, click the **Download Installation media** button on the right.



In the popup window, from the **Options** dropdown select the Digital Ocean (amd64) option. This is a machine image designed for use with Digital Ocean droplets.



**If you are using a different provider than Digital Ocean you may want to choose a different option (or the ISO option for bare metal deployments).**



### Download Installation Media

Talos Version: 1.9.2 ▾

Options: Digital Ocean (amd64) ▾


Pre-Install Extensions

NAME

☐ amdgpu

☐ amd-ucode

Machine User Labels

new label 

Akamai (arm64)

Azure (amd64)

Azure (arm64)

Banana PI BPI-M64 (arm64)

Digital Ocean (amd64) ✓

Digital Ocean (arm64)

Equinix Metal (amd64)

Equinix Metal (arm64)

GCP (amd64)

We don't need to make any further customizations to the image here, so click **Download**. The approximately 100MB machine image will generate and then download to your local computer, and we will upload it to Digital Ocean in the next lesson.

This image is specific to your Omni account and contains preconfigured credentials that will join your machines with your Omni account on boot.

We're now done in the Omni dashboard. Next we'll add the Omni Service Account to Portainer.

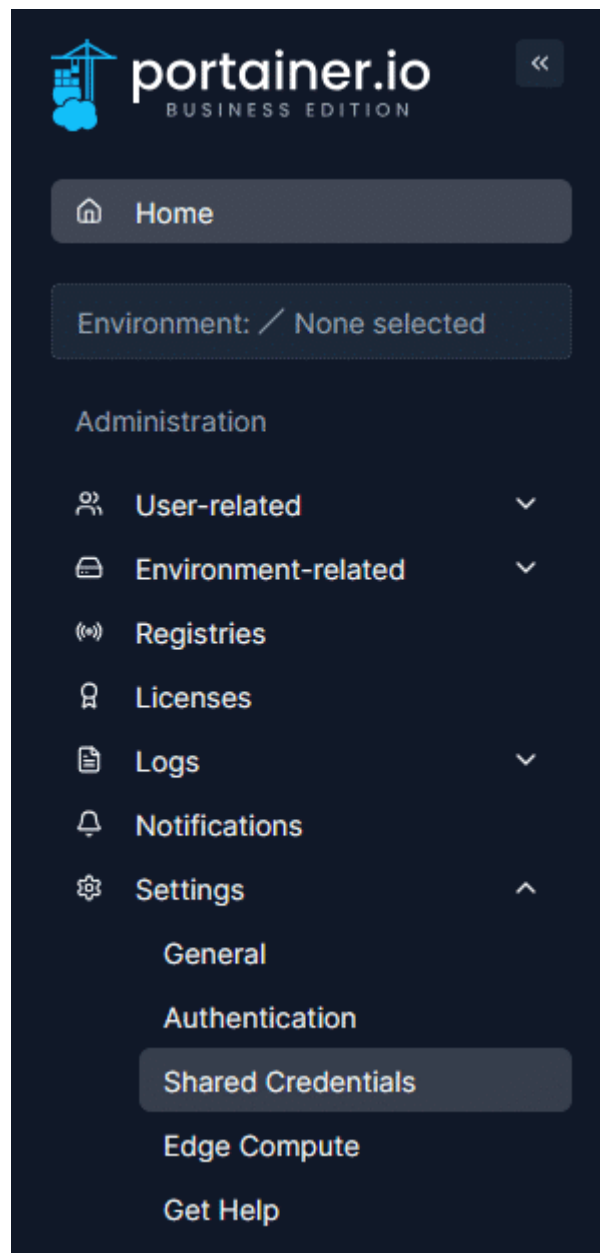
**CONTINUE**



## Adding the Omni Service Account to Portainer

Log into your Portainer server as the administrator. Expand the **Settings** menu on the left and click **Shared Credentials**.










On this fresh install of Portainer you won't have any credential sets configured. Click the **Add credentials** button to create one.



Settings > Shared credentials

## Shared credentials




 admin ▾


 Shared credentials
 
Remove
Add credentials

<input type="checkbox"/>	Name ↓↑	Provider ↓↑
No items available.		


Items per page 10 ▾

Under **Provider**, ensure that Sidero Omni is selected and fill out the form. The **Credentials** name can be any value that identifies this credential set. The **Endpoint URL** should be the value of OMNI\_ENDPOINT we retrieved in the previous step. The **Service Account key** should be the value of OMNI\_SERVICE\_ACCOUNT\_KEY. With these values entered, click **Add credentials**.


Provider




**Sidero Omni**  
Sidero Omni  
Kubernetes  
management  
platform




**Civo**  
Civo Kubernetes




**Akamai Connected Cloud**  
Linode Kubernetes  
Engine (LKE)




**DigitalOcean**  
DigitalOcean  
Kubernetes (DOKS)




**Google Cloud**  
Google Kubernetes  
Engine (GKE)



**Amazon Web Services (AWS)**  
Elastic Kubernetes  
Service (EKS)




**Microsoft Azure**  
Azure Kubernetes  
Service (AKS)



**SSH**  
Provision a  
Kubernetes cluster  
and install Portainer  
using SSH


**Credential details**

 You will need to create an Omni Service Account with an **admin role**. You can create one through the Sidero Omni UI or using omnictl. See our [documentation on creating Sidero Omni credentials](#) for more information.


Any credentials that you set up will be usable by all admin users (although the actual values themselves cannot be viewed).

Credentials name

Omni

Endpoint URL 

https://jamesportainer.omni.siderolabs.io

Service account key 

```

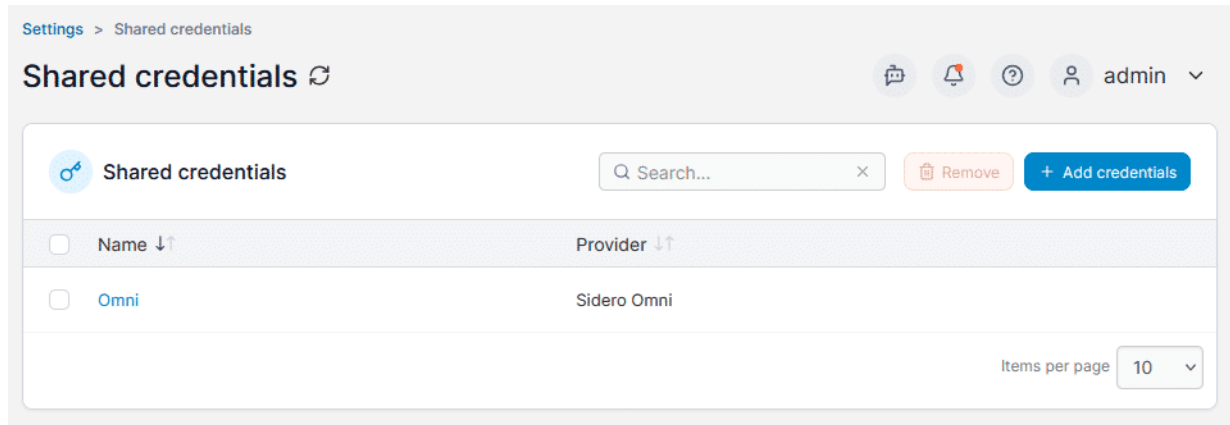
eyJuYW11IjoiUG9ydGFpbmVYliwicGdwX2tleSI6i0tLS0tQkVHSU4gUEdQIFBSSVZBVEUgS0VZIEJMT0NLLS0tLS1cbXueFZnRVo5dFpFaFIKS3
dZQk.JBSGFSdzhCQVfKQTFzQlhiZ0ILdUdBNladkVaS3U2TXVfV3hcnVvOWVoXG5ZdC9nTHNjMVhEOEFBUURmZW5rMFUxUnNPSmNBO
TBmQWFOQzZBQytwdmI2QzhYMG5NakVPRnR3NVF3MmxcbnpTbzhVRzI5ZEdGcGJtVnIRSESeY25acFkyVmhmZMk5ZFc1MEtOXRiWt1YzJ
sa1pYSnZMbVJsZG03Q1xua2dRUUZnb0FSQVdDWWJlOWVnV0pBZUV6Z0FRTENRYOjDwKNOVS8xRERGa1ZUQU1WQ0FVRUZNsUJBQWJaXG5
BUUtIQXdJZUFSWWhCTGJ4NFFuQytzVWRldS9IR28xVC9VTU1XUjZNUFQYUjZnRctJNjIJSUWtVUFZqMWMicbKvUXY5MW1vQXNGUQU4Q
UMrN0t4U3B1T1IISVDN5dVFCQUpPSjN4MEVsUDIRQ1QrN2hFb1diY3FkQzFLV1xuTmFWMHdGQmIxQINDU3lwQXgxMEVaOXRaRWhJS0t3W
UJCQUdYVlFRkFRRUhRRmVmSk5EOTZ6MmWdhaGhYXG4zcn05d2R4WjdCYitpUjg3akpMMVdkbVMZmNGQXdFSUJ3QUeVMOpvb1A3V0Q

```

Add credentials



Your Omni credential set should now appear in the list of credentials.



**CONTINUE**

5

## Summary

In this lesson we have:

- ☐ Created a new Omni account.
- ☐ Created an Omni Service Account for use with Portainer.
- ☐ Downloaded our personalized Digital Ocean machine image for our cluster machines.





Added our Omni Service Account to Portainer's shared credentials.

Next up, we'll use that machine image to provision our cluster machines.



# Create your Talos machines



James Carppe

---

With our Digital Ocean and Omni accounts set up, our Portainer manager server deployed and configured, we're now ready to start creating our cluster. The first step in doing so is to create our machines.

In this lesson we will:

1

Upload our personalized Talos machine image to Digital Ocean.

2

Spin up three droplets to act as our Kubernetes cluster machines.

**START**

---

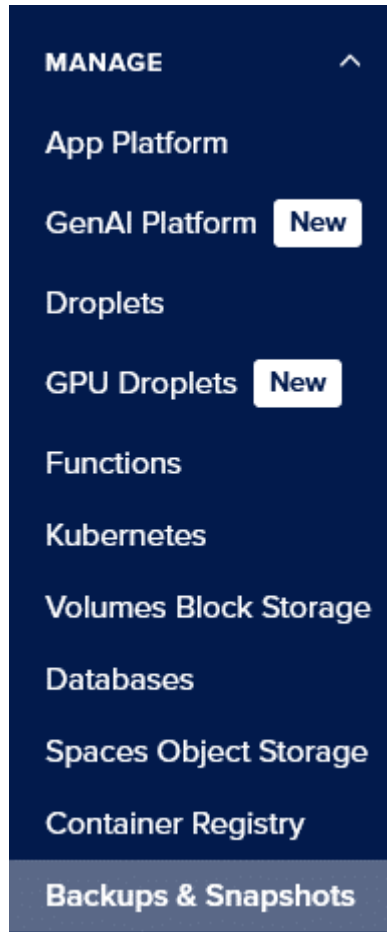
1

## Uploading the machine image



In the previous lesson we downloaded a Digital Ocean specific machine image to use for our Kubernetes nodes. Now we need to upload that image to our Digital Ocean account.

Log into Digital Ocean and expand the **Manage** menu on the left, then click **Backups & Snapshots**.



From here, select the **Custom Images** tab and click **Upload image**.



## Custom Images

[Learn](#) 

Import via URL

Upload Image

Locate the image you downloaded from the Omni dashboard and select it. You'll then be asked to answer a few questions about the image.

You can edit the **image name** if you prefer to have a more user-friendly name, or leave it as the same as the filename. For **Distribution** you can set it as Unknown. For the **datacenter region**, choose the region and datacenter you want to provision the cluster in.



# Upload an Image



## EDIT IMAGE NAME

digital-ocean-amd64-omni-jamesportainer-v1.9.2.raw.gz✓




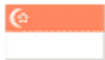





Image Size: 104 MB

## DISTRIBUTION

Unknown

## Choose a datacenter region

Your Image will be located in a single datacenter, but can be transferred later.

<div> New York</div> <div><div>1</div><div>2</div><div>3</div></div>	<div> San Francisco</div> <div><div>1</div><div>2</div><div>3</div></div>	<div> Amsterdam</div> <div><div>2</div><div>3</div></div>
<div> Singapore</div> <div><div>1</div></div>	<div> London</div> <div><div>1</div></div>	<div> Frankfurt</div> <div><div>1</div></div>
<div> Toronto</div> <div><div>1</div></div>	<div> Bangalore</div> <div><div>1</div></div>	<div> Sydney</div> <div><div>1</div></div>




When the form is complete click the **Upload** button. The image will now upload to Digital Ocean's servers and become available in the list of custom images. It may take a few minutes for the upload to complete and for the image to then be distributed to the region you selected.

[Snapshots](#) [Backups](#) [Custom Images](#)

Custom Images [Learn](#)

Import via URL

Upload Image

Image Name	Regions	Uploaded ▲	Tags	Notes
 <b>digital-ocean-amd64-omn...</b> Unknown OS	LON1	2 minutes ago		<a href="#">More ▼</a>

Next we'll provision some servers using this image.

CONTINUE

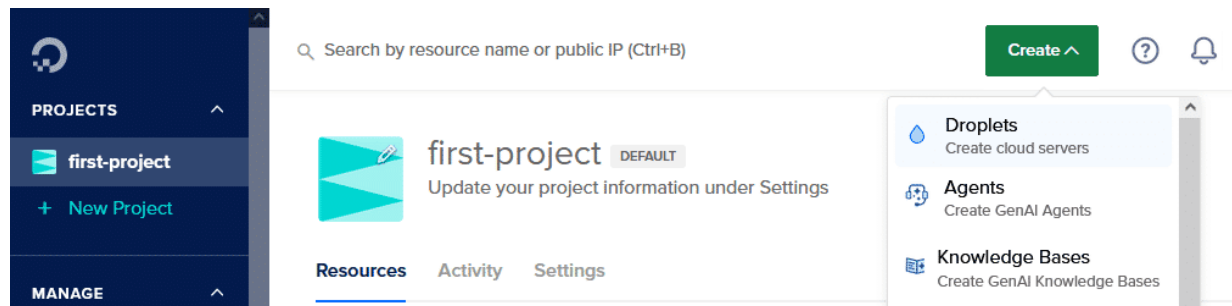
2

## Creating the Talos machines

We can now use the Digital Ocean image to provision the machines we'll use for our Kubernetes cluster. Using the image, the machines will authenticate with your Omni account and become available for cluster provisioning.












In Digital Ocean, click the **Create** button and choose **Droplets**.



Choose the **region** and **datacenter** you uploaded the custom image to.

### Choose Region

 New York	 San Francisco	 Amsterdam
 Singapore	 London	 Frankfurt
 Toronto	 Bangalore	 Sydney

### Datacenter

London • Datacenter 1 • LON1

💡 **Tip: Select the datacenter closest to you or your users** [Dismiss](#)

Avoid any potential latency by selecting a region closest to you - a region is a geographic area where we have one or more datacenters.

VPC Network - **default-lon1** DEFAULT

All resources created in this datacenter will be members of the same VPC network. They can communicate securely over their Private IP addresses.



Scroll down to **Choose an image** and click the **Custom images** tab. You should see your custom image listed - select it.

## Choose an image

OS Marketplace (283) **Custom images**

---

digital-ocean-amd6...	Unknown OS
-----------------------	---------------

For the droplet size we'll again stick with the Basic shared CPU, the Regular disk and the 2GB / 2 CPU option.



## Choose Size

Need help picking a plan? [Help me choose](#) 

### Droplet Type

SHARED CPU	DEDICATED CPU			
<b>Basic</b> (Plan selected)	General Purpose	CPU-Optimized	Memory-Optimized	Storage-Optimized

Basic virtual machines with a mix of memory and compute resources. Best for small projects that can handle variable levels of CPU performance, like blogs, web apps and dev/test environments.


### CPU options

☒ **Regular**  
Disk type: SSD

☐ **Premium Intel**  
Disk: NVMe SSD

☐ **Premium AMD**  
Disk: NVMe SSD

\$4/mo \$0.006/hour	\$6/mo \$0.009/hour	\$12/mo \$0.018/hour	<b>\$18/mo</b> <b>\$0.027/hour</b>	\$24/mo \$0.036/hour	\$48/mo \$0.071/hour
512 MB / 1 CPU 10 GB SSD Disk 500 GB transfer	1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer	2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer	2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer	4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer	8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer

Show all plans

The Talos images require using a **SSH key** for authentication, so if you don't already have one set up from earlier you should do so now. If you already have one, select that.



Choose Authentication Method ?

**SSH Key**

☒ Connect to your Droplet with an SSH key pair

**Password**

☐ Connect to your Droplet as the "root" user via password

Choose your SSH keys

☒ james@S...

New SSH Key

Under **Finalize Details**, we want to provision 3 servers so change the **Quantity** to 3 Droplets. Set **Hostnames** for each droplet as well.

Finalize Details

**Quantity**

Deploy multiple Droplets with the same configuration.

-

3 Droplets

+

**Hostname**

Give your Droplets an identifying name you will remember them by.

talos01













talos02

talos03

When you're ready, click **Create Droplet**. The servers will begin provisioning, and once complete will appear in your list of droplets.





## DROPLETS (4)

 talos02	178.128.35.0	 	Upsize	...
 talos03	167.99.203.65	 	Upsize	...
 talos01	167.99.192.84	 	Upsize	...
 manager	68.183.41.75	 	Upsize	...

Once your machines have completed provisioning, there is one last piece of information we need to gather - the **Private IP** range. We'll need this for some Digital Ocean specific configuration adjustments in the next step.


From the list of droplets, click the name of one of your newly-provisioned machines. In the details for the machine, note down the **Private IP** that is listed. In my case, this is `10.106.0.4`.

[← Back to Droplets](#)

 **talos01**  
in  first-project / 2 GB Memory / 60 GB Disk / LON1 - Unknown OS digital-ocean-amd64-o...

Upsize Droplet

ON

ipv4: 167.99.192.84    ipv6: Disabled    Private IP: 10.106.0.4    Reserved IP: [Enable now](#)    Console: 

Once you have this, we're done in the Digital Ocean control panel.

[CONTINUE](#)



## Summary

In this lesson we:



Uploaded our custom image to Digital Ocean.



Provisioned three droplets using our custom image to act as our cluster nodes.

We're now ready to create our cluster.



# Create your Kubernetes cluster



James Carppe

---

With all our preparation work done, we're now ready to create our Kubernetes cluster!

In this lesson we will provision our environment from Portainer on our Digital Ocean machines, with:

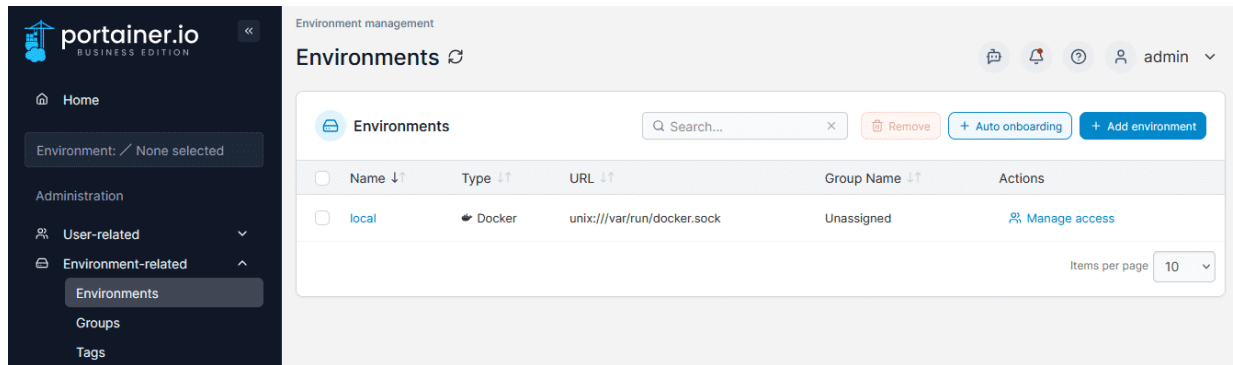
- 1 The Kubernetes metric server enabled.
- 2 Digital Ocean specific configuration adjustments.
- 3 An older version of Kubernetes (to demonstrate a version upgrade).

**START**




## Adding a new environment

Log into Portainer as the local administrator. Expand the **Environment-related** menu on the left and select **Environments**. Here you'll see a list of the environments configured in Portainer - right now you'll only have the `local` environment where Portainer itself is running.



To start provisioning our cluster, click the **Add environment** button. Check the **Create a Kubernetes cluster** option in the Environment Wizard and click **Start Wizard**.




 Environment Wizard

Select your environment(s)

You can onboard different types of environments, select all that apply.


Connect to existing environments



**Docker Standalone**

Connect to Docker Standalone via URL/IP, API or Socket


☐



**Docker Swarm**

Connect to Docker Swarm via URL/IP, API or Socket


☐



**Podman**

Connect to Podman via URL/IP or Socket


☐



**Kubernetes**

Connect to a Kubernetes environment via URL/IP or via kubeconfig import

☐




**ACI**

Connect to ACI environment via API

☐


Set up new environments



**Provision KaaS Cluster**

Provision a Kubernetes cluster via a cloud provider's Kubernetes as a Service

☐



**Create a Kubernetes cluster**


Create a Kubernetes cluster on existing infrastructure

☒

Start Wizard

On the next page, ensure **Talos Kubernetes** is selected. Give your cluster a **Name**. The Portainer server details should be automatically populated with the correct values.


#### Create a Kubernetes cluster



**Talos Kubernetes**

Omni Kubernetes management platform, simplifying the creation and management of Kubernetes clusters

☒



**MicroK8s**

Lightweight Kubernetes


☐

**Beta feature** - so far, Omni cluster management has only been tested in a limited set of scenarios.

This will allow you to create a Kubernetes cluster using Omni with your own existing Talos nodes, and will then deploy the Portainer agent to it.

Name\* 

#### Portainer server details

Portainer API server URL\* 

Portainer tunnel server address\* 



In the **Omni cluster summary** section, your Omni credential set should be automatically selected. For the **Talos version** we can keep it at the default value, but for **Kubernetes version** we'll change this to an older version - for this example, let's choose v1.31.7.

#### Omni cluster summary

Credentials ?	Omni
Talos version ?	v1.9.5
Kubernetes version ?	v1.31.7

In **Cluster machines**, we can select the machines we want to use. You'll see the machines we provisioned in the previous step listed in each dropdown.

For this cluster we're going to use one control plane and two workers, so make your selections accordingly.

#### Cluster machines

① To add Talos nodes, [download the Omni ISO image](#) and boot your machines using this image. For more information, see the [Omni documentation](#).

Control Plane ?	talos01
Main worker pool	talos02 talos03

When provisioning a cluster you can make customizations to the base configuration on both an individual machine and a whole cluster basis. In our case we're going to make a cluster-wide change, and we can do that by expanding the **Cluster configuration patch** section.

In the resulting text box, paste the following block of code:



```
cluster:
  network:
    cni:
      flannel:
        extraArgs:
          - --iface-can-reach=10.106.0.1
        name: flannel
    extraManifests:
      - https://raw.githubusercontent.com/alex1989hu/kubelet-serving-cert-app
      - https://github.com/kubernetes-sigs/metrics-server/releases/latest/download
machine:
  kubelet:
    extraArgs:
      rotate-server-certificates: true
```

Adjust the value of the `--iface-can-reach` option to suit the **Private IP** you copied from the machines you provisioned in the previous step. This is a configuration adjustment that is required for cluster networking to function properly on Digital Ocean. The other changes in the patch enable the necessary functionality for the Kubernetes metrics server to run.



## Cluster Configuration patch

### Edit Cluster Configuration patch

Ctrl+F for search ⓘ

Cluster Configuration Patches allow you to customize settings for all nodes in a cluster or specific groups (such as control plane or worker nodes). See the [Talos Config Reference](#) for more information.

 Copy to clipboard






```
1 cluster:
2   network:
3     cni:
4       flannel:
5         extraArgs:
6           - --iface-can-reach=10.106.0.4
7         name: flannel
8     extraManifests:
9       - https://raw.githubusercontent.com/alex1989hu/kubelet-serving-cert-approver/main/deploy/standalone-install.yaml
10      - https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
11 machine:
12   kubelet:
13     extraArgs:
14       rotate-server-certificates: true
```






When you're ready, click **Provision environment**. You should get a success notification in the top right - when you do, you can click **Close** to exit the Environment Wizard.





You'll be returned to the **Environments** page, where you should now see your cluster provisioning.


Environment management

## Environments

    admin 

 Environments    Remove  + Auto onboarding  + Add environment

<input type="checkbox"/>	Name ↓↑	Type ↓↑	URL ↓↑	Group Name ↓↑	Actions
<input type="checkbox"/>	local	 Docker	unix:///var/run/docker.sock	Unassigned	 Manage access
<input type="checkbox"/>	talos-cluster	 Kubernetes	 Provisioning Omni cluster ⓘ	Unassigned	-


Items per page 10 







Depending on the size of your cluster and the machine selections you made, the cluster provision can take a few minutes to complete. Once it has, the Environments list will update to display the Portainer URL and additional options for the environment.

Environment management

## Environments

 Environments  Remove + Auto onboarding + Add environment


<input type="checkbox"/>	Name ↓↑	Type ↓↑	URL ↓↑	Group Name ↓↑	Actions
<input type="checkbox"/>	local	 Docker	unix:///var/run/docker.sock	Unassigned	 <a href="#">Manage access</a>
<input type="checkbox"/>	talos-cluster	 Kubernetes	https://68.183.41.75:9443	Unassigned	 <a href="#">Manage access</a>

Items per page 10

If you return to the Portainer home page, you should now see the environment listed.



Environments

## Home

 Environments  Refresh Kubeconfig

Click on an environment to manage



Platform Connection... Status Tags Groups Agent Versi... Clear all Sort By ↓↑

**local**  2025-03-20 17:06:31 Standalone 28.0.1 /var/run/docker.sock

Group: Unassigned No tags Local

0 stacks 1 container 1 volume 1 image 2 CPU 2.1 GB RAM

Browse snapshot Live connect Disconnected

**talos-cluster**  2025-03-20 17:10:30 TalosVersion 1.9.5 Running Kubernetes API ready Control plane ready

Group: Unassigned No tags Edge Agent Standard 2.27.2

No snapshot available

Browse snapshot Live connect Disconnected

Items per page 10



## Summary

Congratulations, you have provisioned a three node Kubernetes cluster through Portainer and Omni! Our cluster includes:



The Kubernetes metrics server enabled through a cluster configuration patch.



Some tweaks to the configuration for Digital Ocean.



Kubernetes v1.31.7 deployed so that we can upgrade it later.

Technically you are now ready to go with a working Kubernetes cluster. But next, we'll make some additional tweaks to improve security.



# Secure your cluster



James Carppe

---

Now that we have our cluster up and running, there are some configuration changes we should make to secure it. Omni clusters are quite secure out of the box, but there are still adjustments that can be made.

In this lesson we will:

1

Restrict access to the default namespace.

2

Configure resource quotas on the cluster.

3

Examine the options available via OPA Gatekeeper.

**START**



## Restricting access to the default namespace

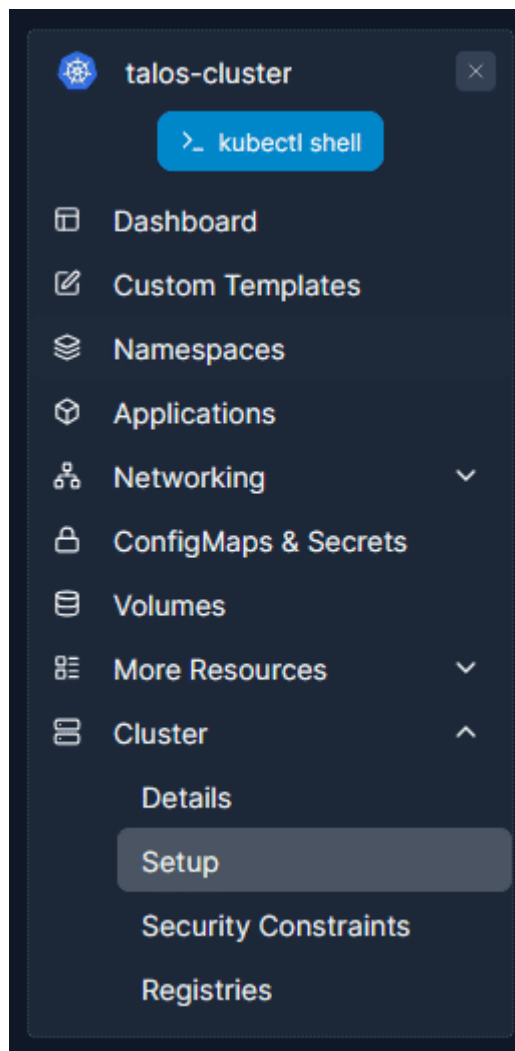
Kubernetes clusters ship with a default namespace configured, which is where workloads that do not specify a namespace would be deployed. For organizational purposes as well as security reasons, we recommend creating namespaces for your workloads.

Namespaces can be configured with access restrictions and resource quotas (both of which we'll cover), and in addition can make it easier to keep track of workloads.

To avoid the possibility of your users deploying to the default namespace (which cannot be configured with resource quotas or access restrictions), we can restrict the usage of it to administrators only.

Log into Portainer as the local administrator, then select your Kubernetes cluster. Expand the **Cluster** menu on the left and select **Setup**.





This page lets you configure your cluster settings to suit your needs. You can find a full description of the options here [in our documentation](#). For now, scroll down to the **Security** section and enable **Restrict access to the default namespace**.

## Security

ⓘ By default, all the users have access to the default namespace. Enable this option to set accesses on the default namespace.

Restrict access to the default namespace ☒

Restrict secret contents access for non-admins (UI only) ⓘ ☐



Click the **Save configuration** button to apply your changes.

CONTINUE

2

## Configuring resource quotas


Limiting the amount of CPU and memory that can be consumed by individual applications and namespaces is a good idea in many cases. Doing so can help to avoid situations where a deployment starts to consume all the available resource on a cluster, causing other deployments to grind to a halt. In addition, Kubernetes by default allows you to over-commit resources when deploying, letting you assign more CPU and memory than is actually available in your cluster to namespaces. For most production workloads, we recommend turning this option off.

First, we'll need to configure resource quotas on our existing namespaces. With your Kubernetes environment selected, click on **Namespaces** in the left menu. This will take you to the list of namespaces on your environment.





Namespaces

## Namespace list



Remove
+ Add with form
+ Create from code

 System resources are hidden, this can be changed in the table settings

<input type="checkbox"/> Name ↓↑	Status ↓↑	Quota ↓↑	Created ↓↑	Actions
<input type="checkbox"/> default	Active	-	2025-03-20 17:04:53	-
<input type="checkbox"/> kubelet-serving-cert-approver	Active	-	2025-03-20 17:06:58	 <a href="#">Manage access</a>

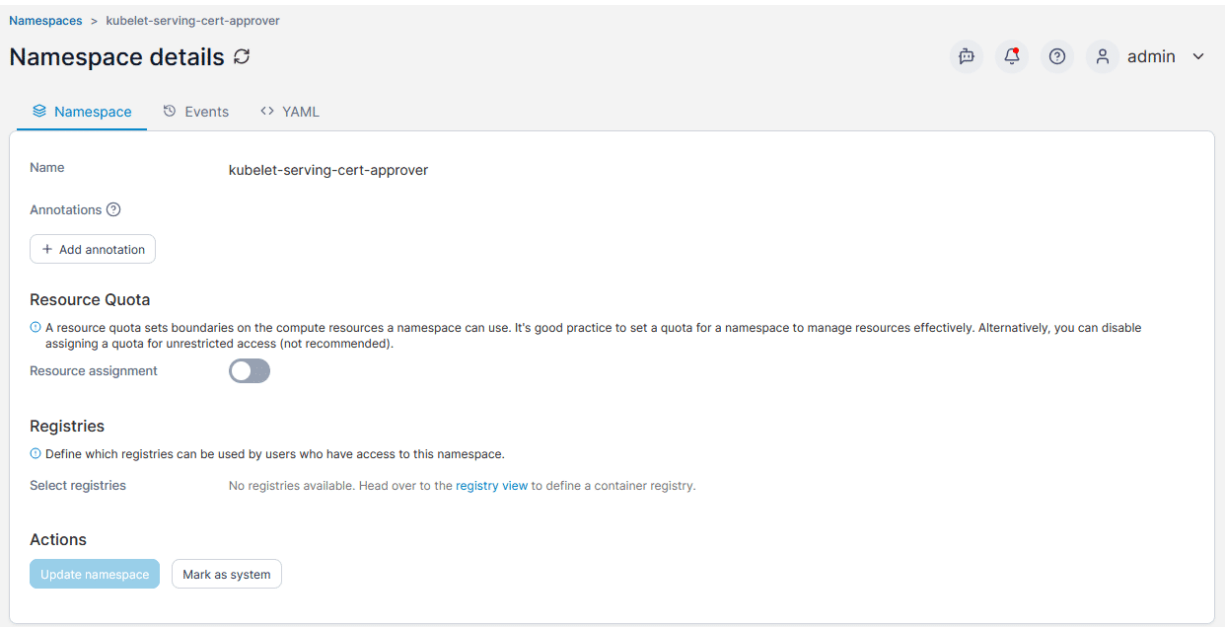
Items per page 10

At this point you should have only two namespaces:

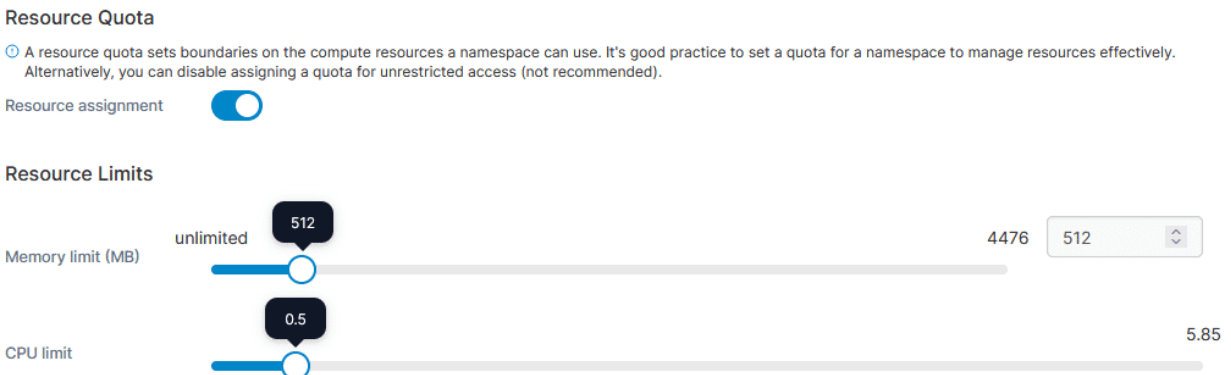
- The default namespace, which we can't adjust quotas on but, thanks to the option we just set, cannot be provisioned to by non-administrator users.
- A namespace called kubelet-serving-cert-approver, which contains the deployment used to rotate the certificates used with the metrics server component we added with our customizations during the deployment.

We want to set a quota on this second namespace, so click on kubelet-serving-cert-approver to be taken to the **Namespace details** page.





Here you can see the details for this namespace. Under **Resource Quota**, enable the **Resource assignment** toggle. This will display two new options for Memory limit (MB) and CPU limit. For this resource, we're going to set these to values of 512 MB for memory and 0.5 for CPU. You can use the sliders to set these values or for memory, type it in manually.





With these values set, click the **Update namespace** button to save your changes, and click **Update** in the popup box to confirm. If you now return to the Namespaces list, you can see that the namespace is now listed as having quota enabled.

Namespaces				
<input type="text" value="Search..."/>				
<span>Remove</span> <span>+ Add with form</span> <span>+ Create from code</span>				
<input type="checkbox"/> System resources are hidden, this can be changed in the table settings				
<input type="checkbox"/> Name ↓↑	Status ↓↑	Quota ↓↑	Created ↓↑	Actions
<input type="checkbox"/> default	Active	-	2025-03-20 17:04:53	-
<input type="checkbox"/> kubelet-serving-cert-approver	Active	Enabled	2025-03-20 17:06:58	<a href="#">Manage access</a>
Items per page 10				

Let's now disable resource over-commit across the cluster. With your Kubernetes environment selected, expand **Cluster** in the left menu and click **Setup**, then scroll down to the **Resources and Metrics** section.

#### Resources and Metrics

☐ By **DISABLING** resource over-commit (highly recommended), you can **ONLY** assign namespaces CPU and memory resources that are less (in aggregate) than the cluster total minus any system resource reservation.

☐ By **ENABLING** resource over-commit, you can assign namespaces more resources than are physically available in the cluster. This may lead to unexpected deployment failures if there are insufficient resources to service demand.

Allow resource over-commit



☐ Enabling the metrics feature allows users to use horizontal pod autoscaling and to see container and node resource usage. This requires [metrics server](#) or [prometheus](#) to be running in your cluster. On any subsequent disabling of the feature, existing deployed applications with autoscaling will still autoscale (you would have to remove their autoscaler definitions to stop this).

Enable features using the metrics API



Disable the **Allow resource over-commit** toggle. This will show a new option to specify the **system resource reservation**. This is a percentage of resource (CPU and memory) that Kubernetes will keep in reserve for system functionality. For now we'll leave this on the default of 20%.



Allow resource over-commit



System resource reservation %

When you're ready, click **Save configuration** to apply the changes.

CONTINUE

3

## OPA Gatekeeper

OPA Gatekeeper is a pod security policy application and lets us specify more fine-grained policies on what can and cannot be done on our cluster. For this workshop we'll look at some of the options that are available through OPA Gatekeeper.

With your Kubernetes environment selected, expand the **Cluster** menu on the left and select **Security Constraints**.



## Security constraints

### Pod security constraints

Enable pod security constraints



Save settings

### Pod security constraints

Enable pod security constraints



#### Constraints

Controls whether any container in a pod can enable privileged mode

Restrict running privileged containers



Controls whether the pod containers can share the process ID namespace and host IPC namespace

Restrict host namespace



Controls whether the pod may use networking ports

Restrict host networking ports



Controls which volume sources may be used

Restrict volume types





A full description of each option is available [in our documentation](#). For now, let's enable **Restrict running privileged containers**.

Enable pod security constraints



### Constraints

Controls whether any container in a pod can enable privileged mode

Restrict running privileged containers



Scroll down and click **Save settings** to apply the policy. Note that the first time you set and save an option here it may take a little longer to apply, as in the background Portainer is deploying OPA Gatekeeper on your cluster in order to provide the policies.

CONTINUE

4

## Summary

In this lesson we adjusted the configuration of our Kubernetes cluster to make it more secure. We:





Restricted access to the default namespace to administrators only.



Set resource quotas on our existing namespace and disabled resource over-commit on the cluster.



Looked at the policy options that OPA Gatekeeper provides and restricted running privileged containers on the cluster.

In the next lesson we'll demonstrate how you can update your Kubernetes cluster from within Portainer.



# Upgrade the Kubernetes version



James Carppe

Deploying a Kubernetes cluster with Portainer and Omni gives us some powerful tools when managing the cluster. New versions of Kubernetes are released regularly containing fixes and new features, and you can easily upgrade your cluster's Kubernetes version with a few clicks.

In this lesson we will upgrade our Kubernetes cluster to the latest version.

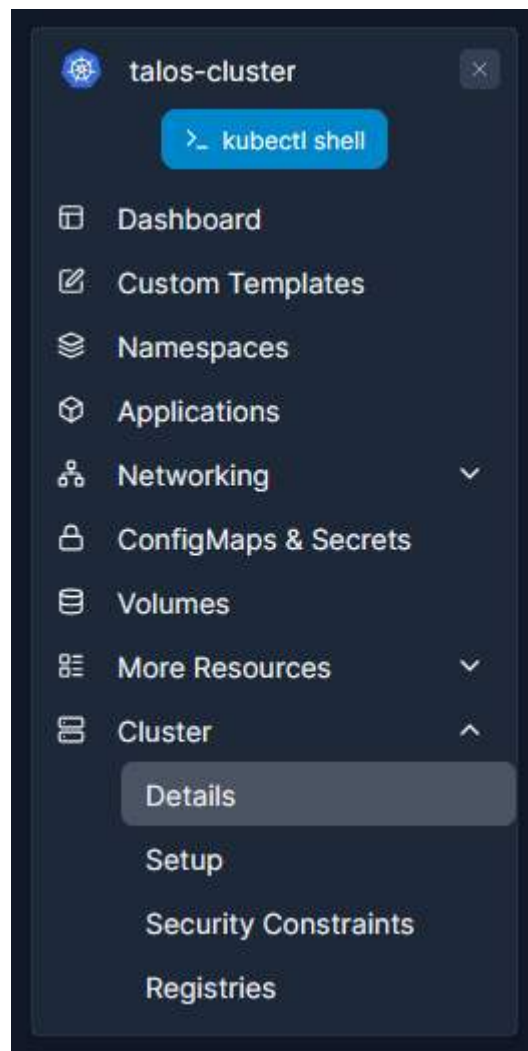
START

1

## Upgrading Kubernetes


Log into Portainer as the local administrator. Select your Kubernetes environment then expand the **Cluster** menu on the left and select **Details**.





On the **Cluster** page you will see information about your cluster including resource reservation and usage, as well as tools to manage your cluster. Scroll down to the **Omni cluster management** section.



 **Omni cluster management**

⚠ Beta feature - so far, Omni cluster management functionality has only been tested in a limited set of scenarios.

Kubernetes version ⓘ

v1.31.7 (current) ▼

ⓘ New Kubernetes version(s) available!

Update Kubernetes version

Talos version ⓘ

v1.9.5 (current) ▼

Update Talos version

Here we can see the current Kubernetes and Talos versions. When we provisioned the cluster we chose an older version of Kubernetes (1.31.7) which is listed as current, and you will note the message indicating a new Kubernetes version is available. If we click the **Kubernetes version** dropdown we can see the options available to us.

v1.31.7 (current) ▼

ⓘ New Kubernetes version(s) available!

v1.31.4

v1.31.5

v1.31.6

v1.31.7 (current)

v1.32.0

v1.32.1

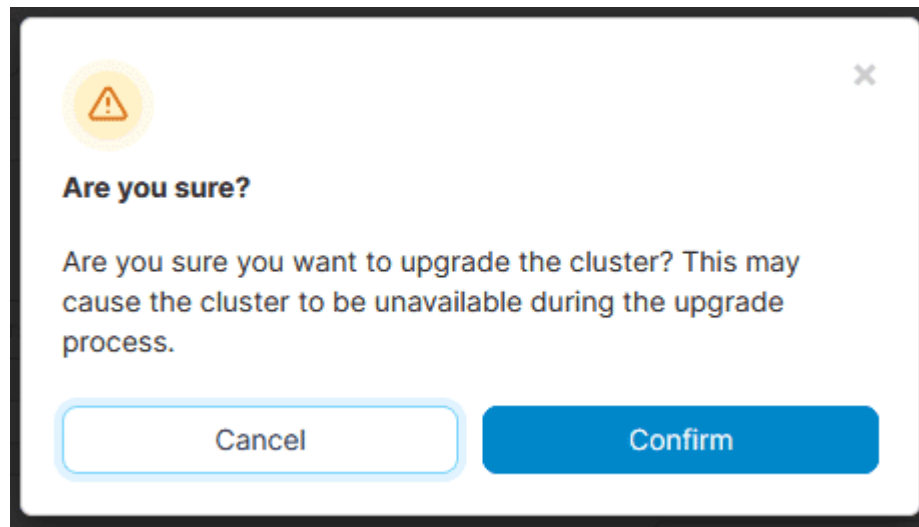
v1.32.2

v1.32.3

Scroll down in the list to find the most recent version of Kubernetes available (in my case, v1.32.3) and select it, then click **Update Kubernetes version**. You'll be shown a warning



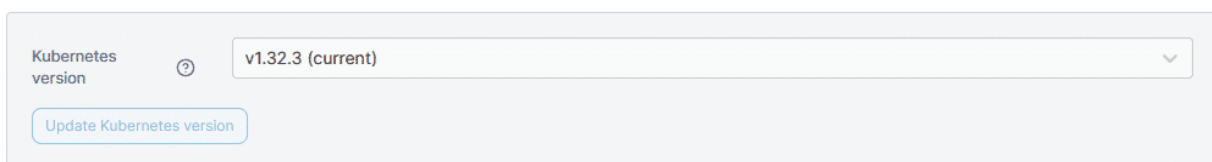
that the cluster may become unavailable during this upgrade - since we're not running any workloads yet, this is fine. Click **Confirm** to proceed.



The upgrade process will now begin. Note that this process may take a few minutes to complete. You'll see status updates with detail on how the upgrade is proceeding.

🕒 Updating Kubernetes version  
pre-pulling images (4/7): pulled "registry.k8s.io/kube-proxy:v1.32.3"

Once the upgrade completes, the current version listed will be updated to the version we selected.





We can confirm this has rolled out to all nodes in the cluster by scrolling down to the **Nodes** section and checking the listed version.

Nodes								
<input type="text" value="Search..."/>								
<div><span>Reboot</span> <span>Remove</span> <span>+ Add nodes</span></div>								
<input type="checkbox"/>	Name ↓↑	Role ↓↑	Status ↓↑	CPU ↓↑	Memory ↓↑	Version ↓↑	IP Address ↓↑	Actions
<input type="checkbox"/>	talos01 <span>api</span>	Control plane	Ready	1.95	1.4 GB	v1.32.3	167.99.192.84	<span>📶</span> <span>📄</span>
<input type="checkbox"/>	talos02	Worker	Ready	1.95	1.5 GB	v1.32.3	178.128.35.0	<span>📶</span> <span>📄</span>
<input type="checkbox"/>	talos03	Worker	Ready	1.95	1.5 GB	v1.32.3	167.99.203.65	<span>📶</span> <span>📄</span>
								Items per page <span>10</span> ▾

CONTINUE

2

## Summary

In this (quick) lesson we successfully upgraded our Kubernetes cluster to the latest version in just a few clicks, without having to touch the command line at all.

In the next lesson we'll start configuring user-level access using Role-based Access Control (RBAC).



# Role-based Access Control (RBAC)



James Carppe

---

Role-based Access Control, or RBAC, is a way in which you can specify the access level of users within your organization to the resources available via Portainer. Users can be provided with roles at both an individual and group level, for environments themselves as well as resources within those environments.

In this lesson we will:

1

Configure our GitHub OAuth user with access to our Kubernetes environment.

2

Create a namespace on the environment and give our GitHub OAuth user access to it.

3

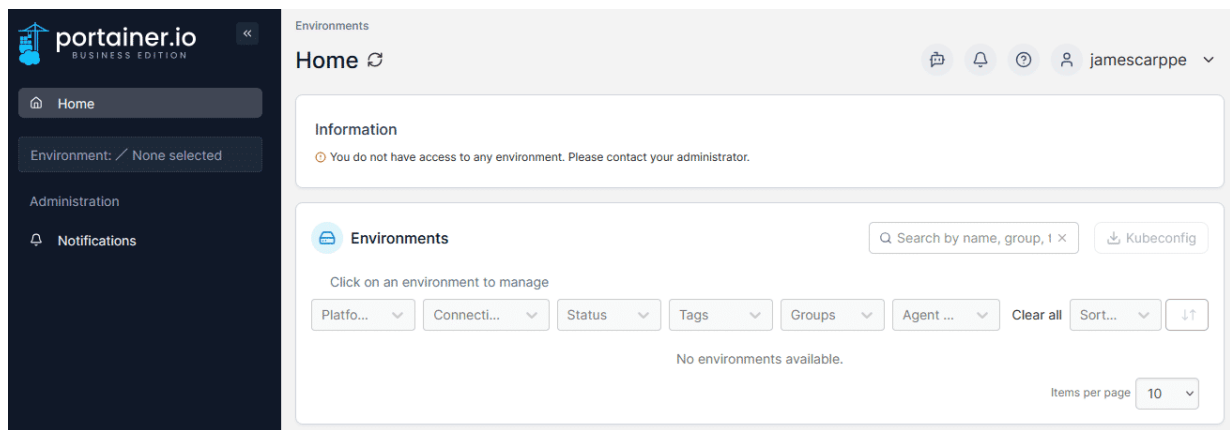
Log in with our GitHub OAuth user and confirm our level of access is as expected.

**START**



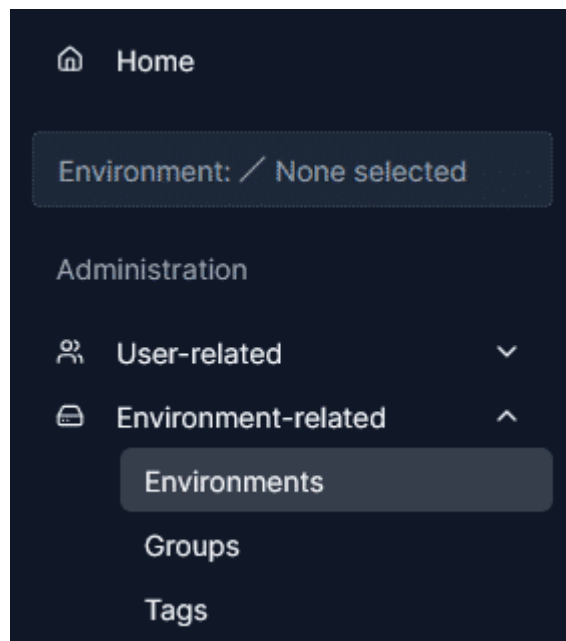
## Giving our user environment access

To start with, we need to give our user access to the new Kubernetes environment. As you'll recall from when we configured OAuth, the GitHub user could not see any environments when logged in.



Let's remedy that. Log into Portainer as the local administrator, then in the left menu expand **Environment-related** and select **Environments**.







You should see our two environments listed - `local` (the local Portainer server environment) and the Kubernetes cluster we created (in my example, this is called `talos-cluster`).


Environments				
<div>Q Search... x Remove + Auto onboarding + Add environment</div>				
<input type="checkbox"/>	Name ↓↑	Type ↓↑	URL ↓↑	Group Name ↓↑
<input type="checkbox"/>	local	Docker	unix:///var/run/docker.sock	Unassigned
<input type="checkbox"/>	talos-cluster	Kubernetes	https://68.183.41.75:9443	Unassigned
				Actions
				<a href="#">Manage access</a>
				<a href="#">Manage access</a>
Items per page 10 v				

Next to your Kubernetes environment, click the **Manage access** link. Here you can manage access to this environment. Scroll down to the **Access** section and you will see that there are no users or groups listed with access to the environment currently (the initial administrator has access to everything so is not listed here).




 **Access**




 Remove

 Update

Access tagged as `inherited` are inherited from the group access. They cannot be removed or modified at the environment level but they can be overridden.

Access tagged as `override` are overriding the group

 Updating user access will require the affected user(s) to logout and login for the changes to be taken into account.


<input type="checkbox"/>	Name 	Type 	Role 
No items available.			


Items per page 10

Scroll back up to the **Create access** section and from the **Select user(s) and/or team(s)** dropdown choose your GitHub user. In the **Role** dropdown, choose the Standard user role.





The Standard user role has full access to resources that they create or are given access to, or that a member of their team has deployed. They do not have any administrative access to either the cluster or to Portainer itself. For more on roles and their respective access levels, [refer to our documentation](#).


 **Create access**

 Adding user access will require the affected user(s) to logout and login for the changes to be taken into account.

Select user(s) and/or team(s)


 jamescarppe





Role


Standard user



+ Create access



When you're ready, click **Create access**. The user will be given the **Standard user** role on the environment and will now appear in the **Access** list.

 **Access**

× Remove Update

Access tagged as **inherited** are inherited from the group access. They cannot be removed or modified at the environment level but they can be overridden.  
Access tagged as **override** are overriding the group

ⓘ Updating user access will require the affected user(s) to logout and login for the changes to be taken into account.

<input type="checkbox"/> Name ↓↑	Type ↓↑	Role ↓↑
<input type="checkbox"/> jamescarppe	user	Standard user <a href="#">Edit</a>

Items per page 10 ▾

CONTINUE

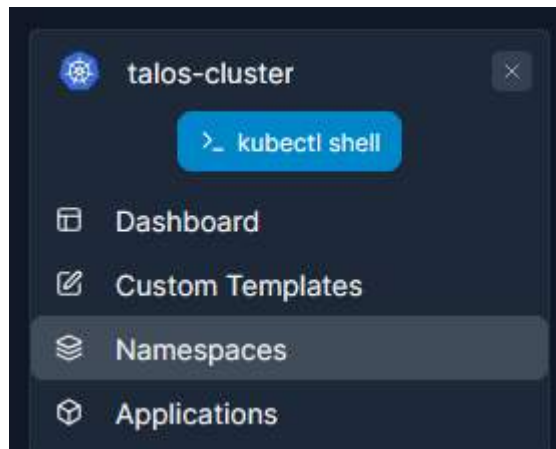
2

## Creating a namespace for the user

Our user now has access to the Kubernetes cluster, but they don't have any namespaces to work in. As an administrator, let's create one for them.

Select your Kubernetes environment then select **Namespaces** from the left menu.





In the **Namespace list** you'll see our existing namespaces. Click on **Add with form** to create a new namespace.

Namespaces

### Namespace list

admin

Namespaces

Remove

Add with form

Create from code

System resources are hidden, this can be changed in the table settings

<input type="checkbox"/>	Name ↓↑	Status ↓↑	Quota ↓↑	Created ↓↑	Actions
<input type="checkbox"/>	default	Active	-	2025-03-20 17:04:53	Manage access

In the **Create a namespace** form, set a **Name** for the namespace. You will also note that we must set memory and CPU limits because we disabled over-commit earlier. Let's give this namespace a memory limit of 2048 MB and a 2 CPU limit.



Name\*

Annotations [?](#)

[+ Add annotation](#)

**Resource Quota**

[?](#) A resource quota sets boundaries on the compute resources a namespace can use. It's good practice to set a quota for a namespace to manage resources effectively. Alternatively, you can disable assigning a quota for unrestricted access (not recommended).

Resource assignment ☒

**Resource Limits**


Memory limit (MB) unlimited 2048 3069

CPU limit unlimited 2 4.18

When you're ready, click the **Create namespace** button. The namespace will be created and you'll be returned to the namespace list, where you'll see your new namespace listed.

<input type="checkbox"/>	kubelet-serving-cert-approver	Active	Enabled	2025-03-20 17:06:58	<a href="#">Manage access</a>
<input type="checkbox"/>	user-namespace	Active	Enabled	2025-03-21 14:43:18 by admin	<a href="#">Manage access</a>

Now we need to give our user access to the namespace. Click on **Manage access** for the namespace we just created. Similar to the environment access management page, this page lists who has access to the namespace (currently nobody but administrators).

 Namespace access


[Remove](#)

<input type="checkbox"/> Name <a href="#">↑↓</a>	Type <a href="#">↑↓</a>
No items available.	

Items per page



Scroll up to the **Create access** section and select your user. You will note that the environment access level for that user is displayed next to their username.

 **Create access**

ⓘ Adding user access will require the affected user(s) to logout and login for the changes to be taken into account.

Select user(s) and/or team(s)


jamescarppe | Standard user

x

⌵

+ Create access

When you're ready, click **Create access**. The user will be given access to the namespace and will appear in the **Namespace access** list.

 **Namespace access**

Q Search...

x

Remove

<input type="checkbox"/> Name ↓↑	Type ↓↑
<input type="checkbox"/> jamescarppe	user

Items per page 10 ⌵

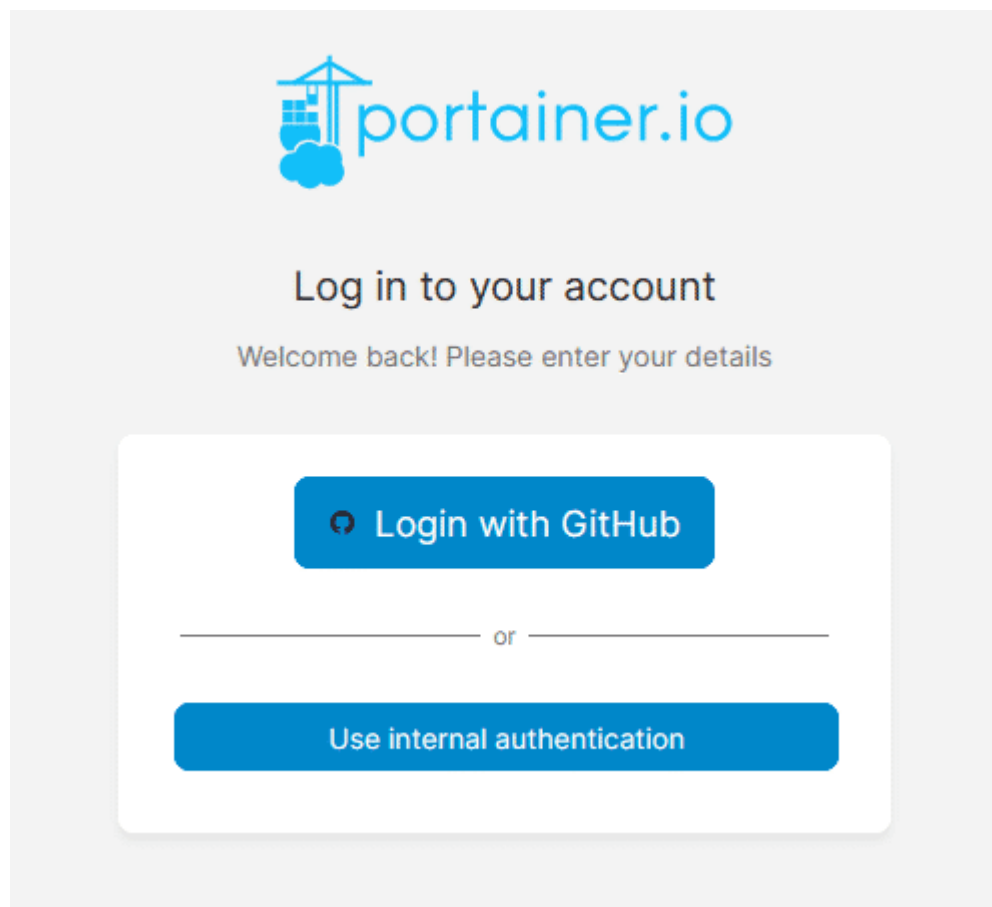
Now let's make sure that all worked, and log in as the GitHub user.

CONTINUE



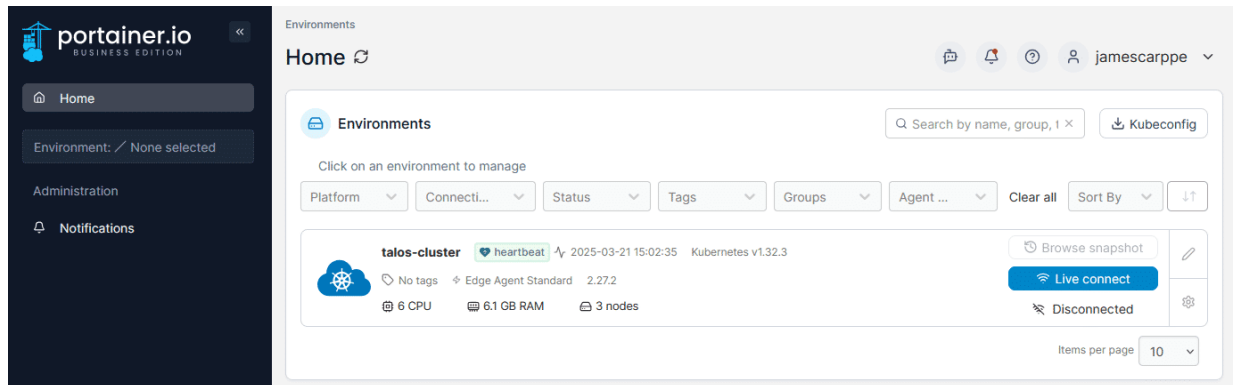
## Testing access

Log out of Portainer (click the arrow next to your username in the top right and select **Log out**) and log back in as the GitHub user by clicking **Login with GitHub** and providing the credentials if necessary.

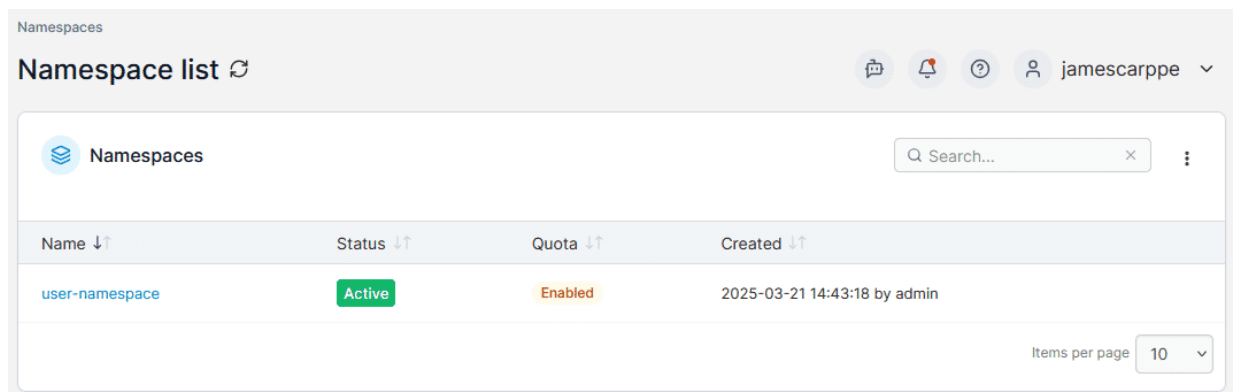


Once you're logged in, you will see you now have access to one environment - our Kubernetes cluster. You'll also note that we don't have the full Administration menu options on the left, as we are a standard user.





Select the environment and then select **Namespaces** in the left menu. In the list of namespaces, you'll see only the namespace we just created.



CONTINUE



## Summary

In this lesson, we:



Gave our user access to the Kubernetes environment as the Standard user role.



Created a namespace for our user and gave them access to it.



Logged in as our user and confirmed we only have access to what we specified.

In the next lesson we'll use this user to deploy an application on the Kubernetes cluster.



# Deploy an application



James Carppe

Our cluster is now configured and we have given our user access to it as a standard user.  
Now let's deploy an application to the cluster.

In this lesson we will:

☐

Deploy an application using GitOps.

☐

Make a change to the application configuration in Git.

☐

Confirm that our update has automatically applied.

**START**



# Deploying an application with GitOps

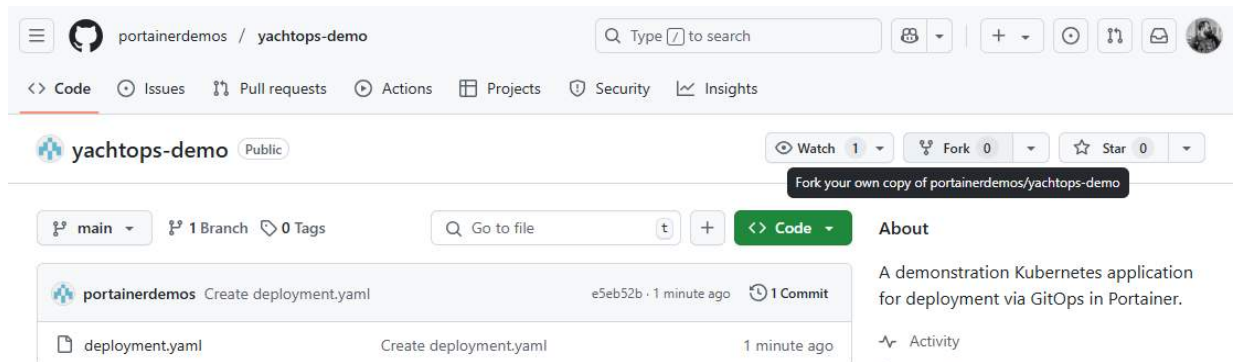
There are a number of methods you can use to deploy an application on a Kubernetes cluster. In this example, we'll demonstrate how you can use a Git repository containing a YAML manifest to deploy an application through Portainer.

Before we start, we're going to fork an example repository that we have created for this purpose. We want to create a fork as we are going to make changes to the repository contents in the next step.

You can find the example repository at the following URL:

```
https://github.com/portainerdemos/yachtops-demo
```

Go to the above repository in a web browser and click the **Fork** button.






Choose from the list of owners to fork to (if you have access to multiple owners) and edit the name of the repo if you so desire, then click **Create fork**.

## Create a new fork

A *fork* is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

*Required fields are marked with an asterisk (\*).*

Owner \*

 jamescarppe ▾

Repository name \*

/ yachtops-demo

✔ yachtops-demo is available.


By default, forks are named the same as their upstream repository. You can customize the name to distinguish it further.

Description (optional)

A demonstration Kubernetes application for deployment via GitOps in Portainer.

☒ Copy the `main` branch only

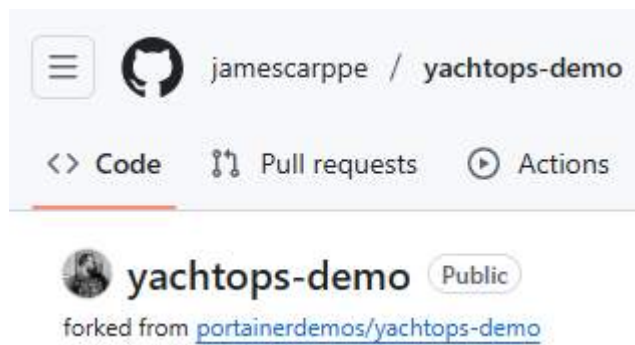
Contribute back to portainerdemos/yachtops-demo by adding your own branch. [Learn more.](#)

 You are creating a fork in your personal account.

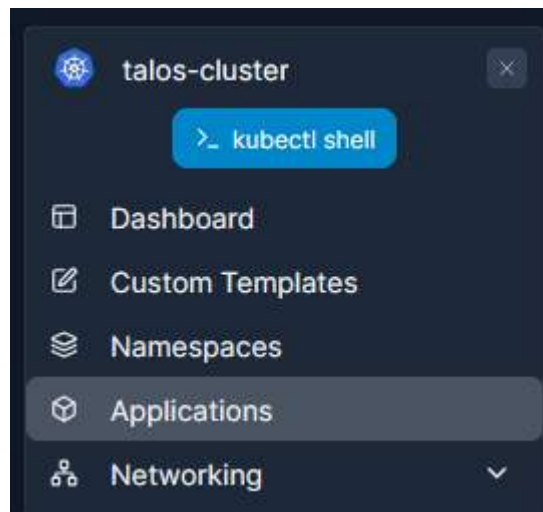
Create fork

Once the fork has been created you'll have a copy of the repo in your own GitHub account. We'll deploy our application from there.



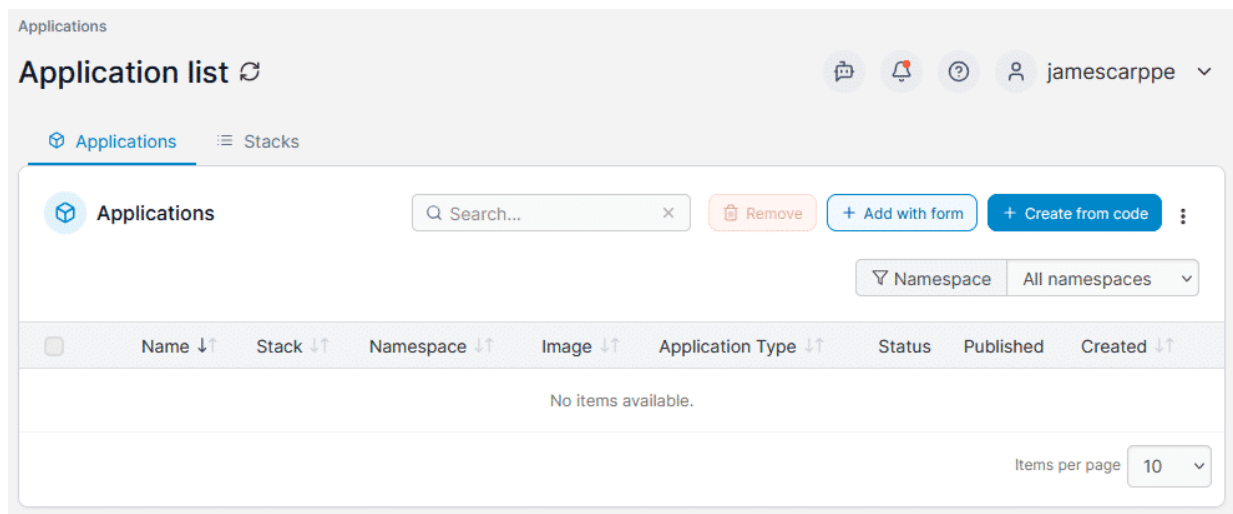


Back in Portainer, log in as your GitHub user and select your Kubernetes environment. Next, click **Applications** in the left menu.



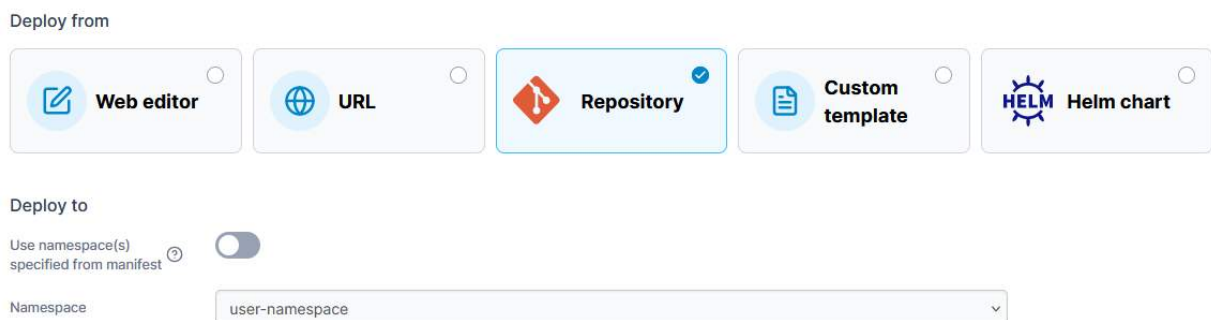
In the **Applications list**, you can see we have no applications currently deployed on the cluster that this user can access. You can use the **Namespace** dropdown to filter the list by namespace, or show all namespaces.





To create our application, click on the **Create from code** button.

On the **Create from code** page, ensure that **Repository** is selected and that the namespace we created earlier is selected in **Namespaces**.




As we're deploying from a Git repository, we need to specify the details. This repository is public, so we don't need to specify authentication. Provide the URL of your forked repository in the **Repository URL** field. The **Repository reference** should automatically populate, and enter deployment .yaml in the Manifest path field.



## Git repository

Authentication



 You can use the URL of a git repository.

Repository URL\*

https://github.com/jamescarppe/yachtops-demo



Skip TLS Verification 



 Specify a reference of the repository using the following syntax: branches with `refs/heads/branch_name` or tags with `refs/tags/tag_name`.

Repository  
reference

\*

refs/heads/main




 Indicate the path to the Manifest file from the root of your repository (requires a yaml, yml, json, or hcl file extension).


Manifest path\*

deployment.yaml

At the bottom of the form, toggle on **GitOps updates**. Here we can configure how Portainer checks for updates to your deployment. Ensure **Polling** is selected as the mechanism and set the **Fetch interval** to 1m.

GitOps updates 



 Any changes to this stack or application that have been made locally via Portainer or directly in the cluster will be overwritten by the git repository content, which may cause service interruption.

Mechanism

Polling

Webhook

Fetch interval\* 



1m






When you're ready, click **Deploy**. Portainer will retrieve the manifest YAML from the Git repository and deploy it on your environment. You should see it appear in the application list, initially with a status indicating 0 of 1 replicas.












Applications


## Application list

 Applications 

 Applications   Remove  + Add with form  + Create from code 

 Namespace All namespaces 



<input type="checkbox"/>	Name 	Stack 	Namespace 	Image 	Application Type 	Status	Published	Created 
<input type="checkbox"/>	nginx-deployment	-	user-namespace	nginx:1.14.2	Deployment	 Replicated 0 / 1	No	2025-03-21 16:06:10 by jamescarppe






Items per page 10 



Once Kubernetes pulls the image and determines where to place it, this will update to 1 of 1 replicas. You can refresh the list with the refresh button in the top left.








Applications


## Application list

 Applications 

 Applications   Remove  + Add with form  + Create from code 

 Namespace All namespaces 

<input type="checkbox"/>	Name 	Stack 	Namespace 	Image 	Application Type 	Status	Published	Created 
<input type="checkbox"/>	nginx-deployment	-	user-namespace	nginx:1.14.2	Deployment	 Replicated 1 / 1	No	2025-03-21 16:06:10 by jamescarppe

Items per page 10 

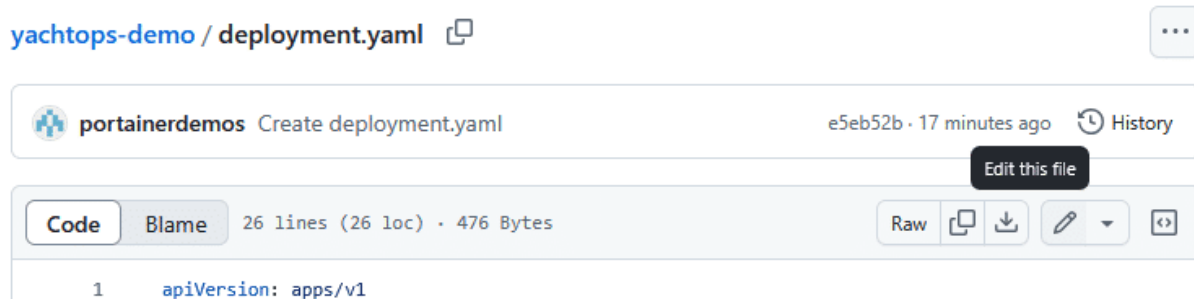
CONTINUE



## Making a change to the application

The application has deployed successfully to your cluster. But let's say we wanted to make a change. With the way we've configured this application, all we need to do is update the manifest in the Git repository and Portainer will automatically pick up the change and update the deployment.

In your fork of the Git repository, make a change to the `deployment.yaml` file. You can do this through the GitHub web interface by selecting the file and clicking on the pencil icon.

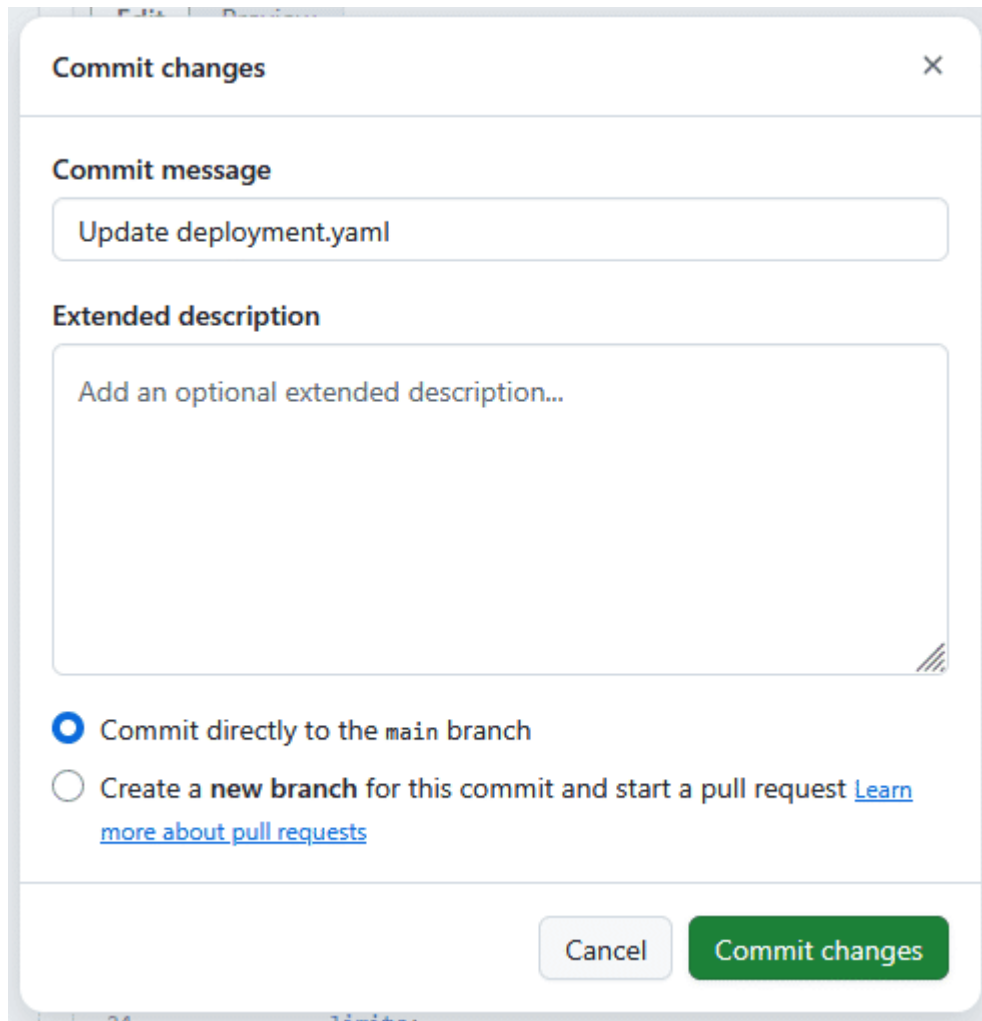


For this example, let's imagine we wanted to increase the number of replicas of our container to 2. On line 9 of the manifest file, change `replicas: 1` to `replicas: 2`.

```
4     name: nginx-deployment
5   spec:
6     selector:
7       matchLabels:
8         app: nginx
9     replicas: 2
10    template:
11      metadata:
```



With this changed, click the **Commit changes** button and click **Commit changes** again in the popup window.



**Commit changes** [X]

**Commit message**

Update deployment.yaml

**Extended description**

Add an optional extended description...

☒ Commit directly to the main branch

☐ Create a new branch for this commit and start a pull request [Learn more about pull requests](#)

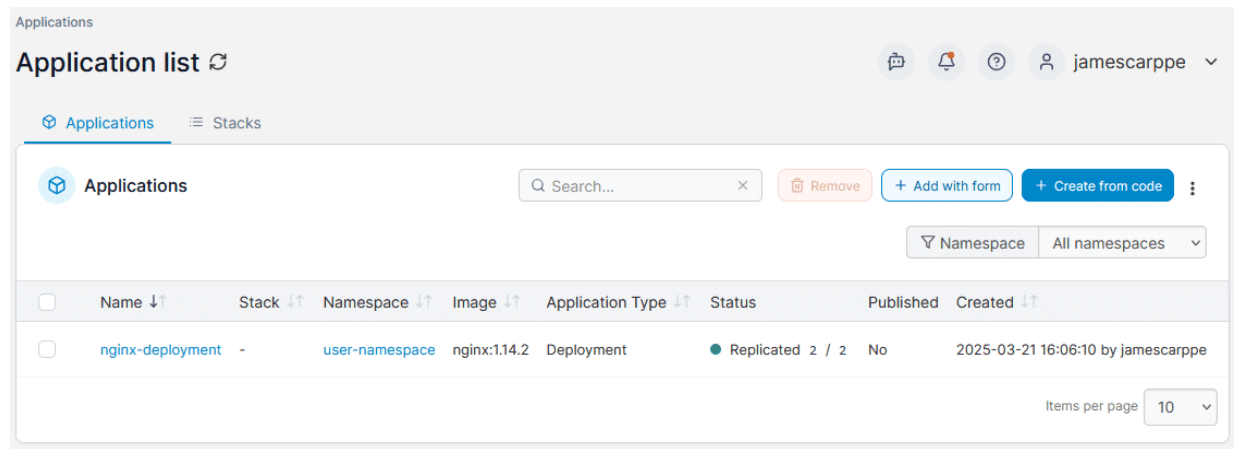
Cancel Commit changes

Your changes will be committed to the repository.

When we created this application in Portainer, we set the polling interval for GitOps to one minute. This means that every minute, Portainer will check the repository for any changes. If it finds any, it will retrieve those changes and apply them to the application on the cluster.



If you return to the **Application list** in Portainer once a minute has passed, you should see the status has changed for the application:



The screenshot shows the Portainer 'Application list' interface. At the top, there's a header with 'Applications' and 'Application list' with a refresh icon. On the right, there are icons for chat, notifications, help, and a user profile 'jamescarppe'. Below the header, there are tabs for 'Applications' (selected) and 'Stacks'. A search bar and a 'Remove' button are present. To the right of the search bar are buttons for '+ Add with form' and '+ Create from code'. Below these is a 'Namespace' filter dropdown set to 'All namespaces'. The main part of the interface is a table with the following columns: Name, Stack, Namespace, Image, Application Type, Status, Published, and Created. There is one application listed: 'nginx-deployment' in the 'user-namespace' with image 'nginx:1.14.2' and type 'Deployment'. Its status is 'Replicated 2 / 2' with a green dot. The 'Published' column shows 'No' and the 'Created' column shows '2025-03-21 16:06:10 by jamescarppe'. At the bottom right, there is a 'Items per page' dropdown set to '10'.

Name	Stack	Namespace	Image	Application Type	Status	Published	Created
nginx-deployment	-	user-namespace	nginx:1.14.2	Deployment	Replicated 2 / 2	No	2025-03-21 16:06:10 by jamescarppe

If you click the name of the application then go to the **Events** tab, you can see the events that occurred to update the replica count.



Application	Placement	Events	YAML
-------------	-----------	--------	------

Events

Q Search... X

Date ↑↓	Kind ↑↓	Type ↑↓	Message ↑↓
2025-03-21 16:16:16	Pod	Normal	Started container nginx
2025-03-21 16:16:15	Pod	Normal	Successfully pulled image "nginx:1.14.2" in 4.381s (4.381s including waiting). Image size: 44710204 bytes.
2025-03-21 16:16:15	Pod	Normal	Created container: nginx
2025-03-21 16:16:11	Pod	Normal	Pulling image "nginx:1.14.2"
2025-03-21 16:16:10	Pod	Normal	Successfully assigned user-namespace/nginx-deployment-59cf44d54c-dgjdj to talos02
2025-03-21 16:16:10	Deployment	Normal	Scaled up replica set nginx-deployment-59cf44d54c from 1 to 2
2025-03-21 16:06:11	Pod	Normal	Container image "nginx:1.14.2" already present on machine
2025-03-21 16:06:11	Pod	Normal	Created container: nginx
2025-03-21 16:06:11	Pod	Normal	Started container nginx
2025-03-21 16:06:10	Pod	Normal	Successfully assigned user-namespace/nginx-deployment-59cf44d54c-8rcgg to talos03
2025-03-21 16:06:10	Deployment	Normal	Scaled up replica set nginx-deployment-59cf44d54c from 0 to 1

Items per page 25

CONTINUE

3

## Summary

In this lesson we:



Forked a Git repository containing an application manifest and deployed the application from our forked repo to our Kubernetes cluster.





Made a change to the manifest in the Git repository.



Watched the change get automatically picked up by Portainer and applied to the application on the cluster.

Next we'll summarize everything we've covered in this course.



# Summary



James Carppe

---

Through the lessons in this course, we have:



Provisioned a management server and deployed Portainer Business Edition.



Configured our Portainer server with external authentication via GitHub OAuth.



Created an Omni account and service account, and configured those credentials in Portainer.



Created three Talos droplets in Digital Ocean for our Kubernetes cluster.



Created a three node Kubernetes cluster in Portainer via Omni.



Configured security settings on the cluster through OPA Gatekeeper and set up resource quotas.



Upgraded the version of Kubernetes to the latest release.





Configured Role-based Access Control for our GitHub user and created a namespace for them to use.



Logged in as our GitHub user and deployed an application using GitOps.



Made a change to the application manifest in the Git repository and watched the application automatically update with the change.

Congratulations! You've managed to get a real, working Portainer and Kubernetes cluster up and running with an application deployed in under two hours!