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.

- **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

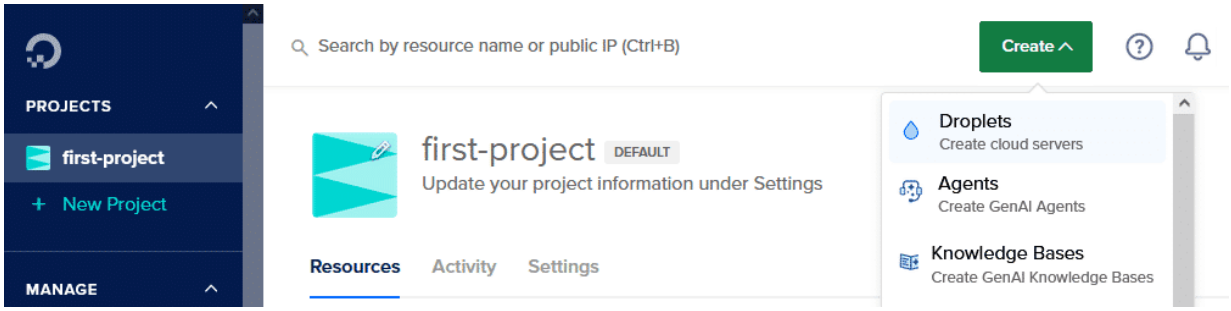
1

Provisioning a server on Digital Ocean

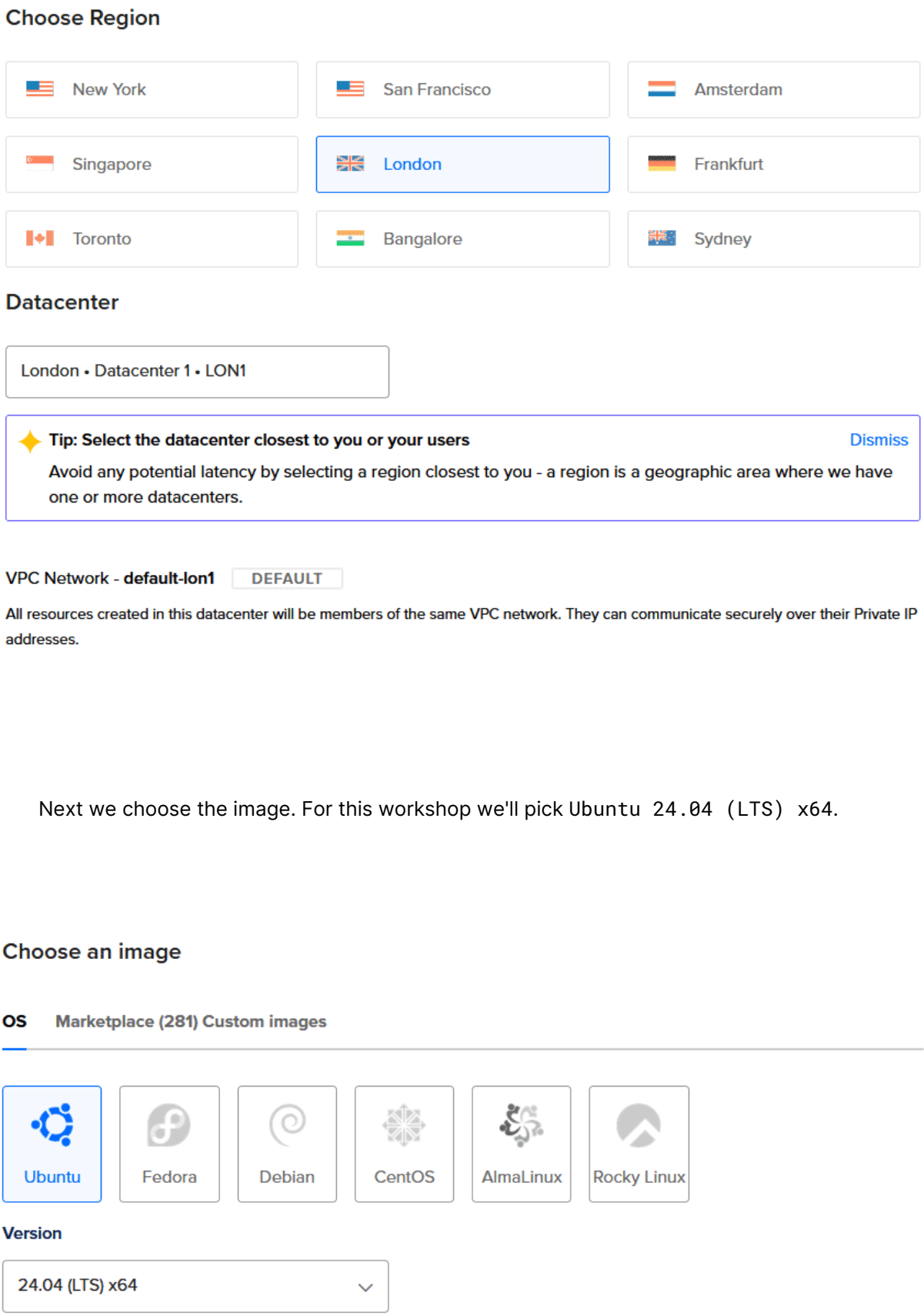
For this workshop, we're going to spin up a Ubuntu Server VM in Digital Ocean to act as our "management server". We'll install Docker on this server, then Portainer itself.

> **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**.

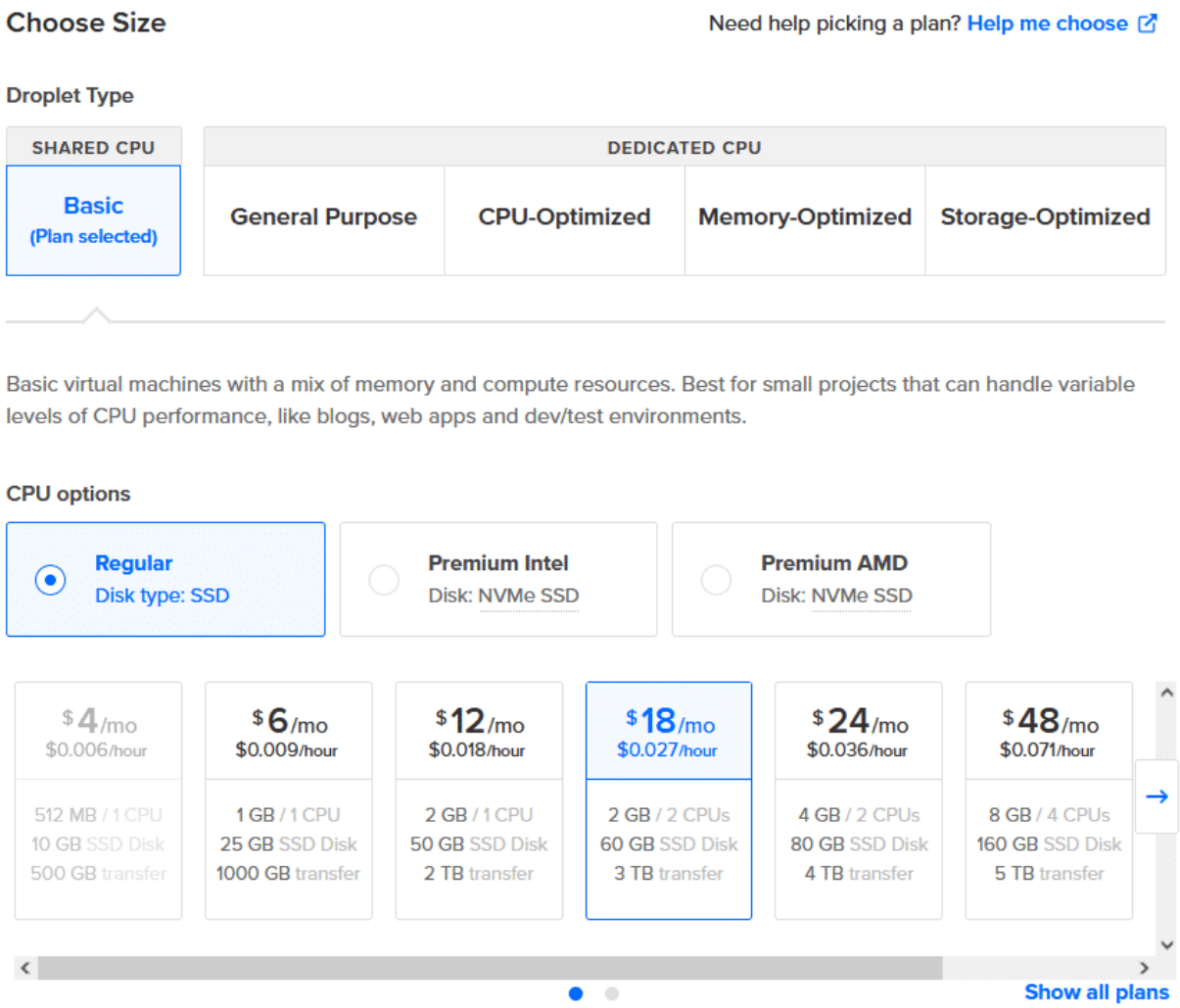


Choose the **region** you want to deploy in and optionally the **datacenter** within that region. Make a note of what you choose as we'll want to use the same region and datacenter when we create the Kubernetes cluster. Also make note of the **VPC Network** - we'll need that too.

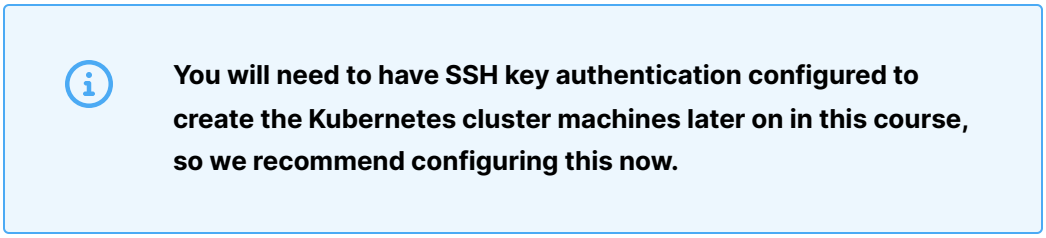


Next we choose the image. For this workshop we'll pick `Ubuntu 24.04 (LTS) x64`.

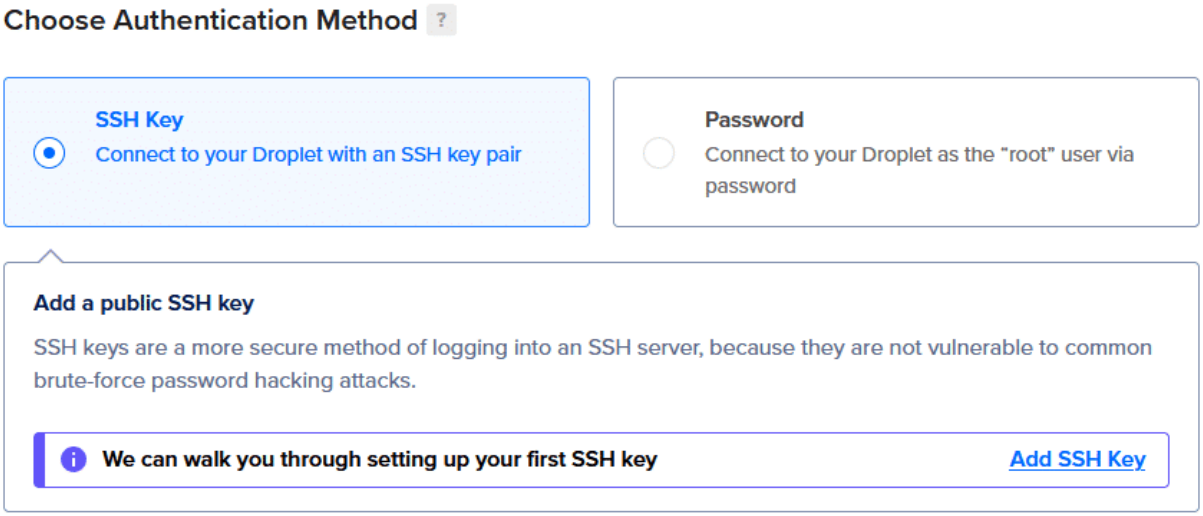
Now we pick the size of the VM. Since this VM is only going to run the Portainer server and not any actual workloads, we can choose a `Basic` type, `Regular` CPU type and the `2GB / 2 CPUs` plan.



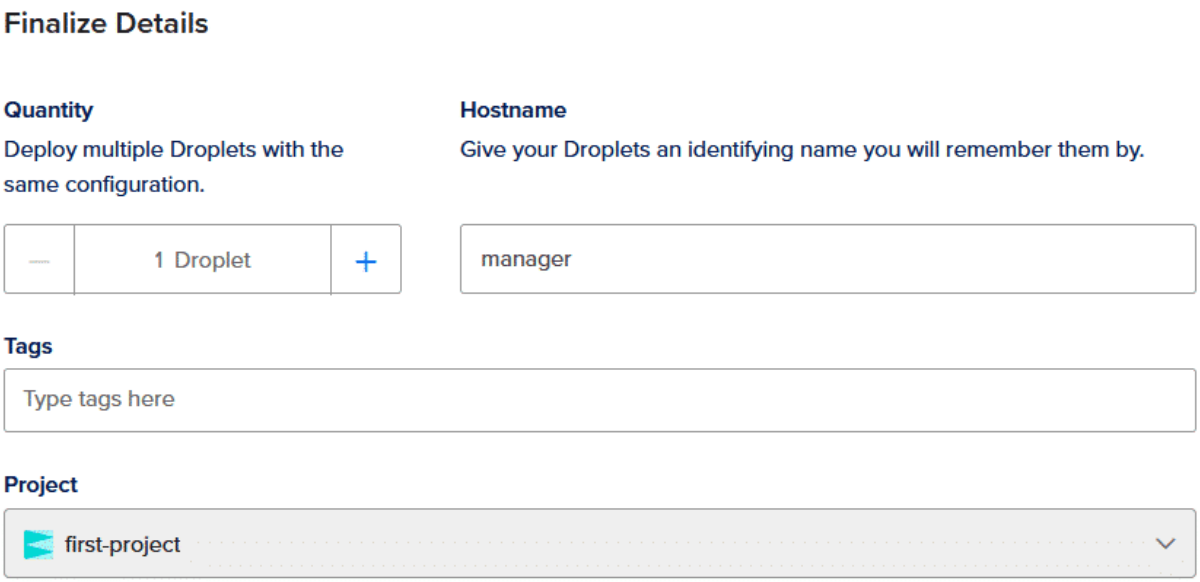
Scroll down to the **Choose Authentication Method** section. Here you can decide how you'll access this server for the initial setup. If you have a SSH key pair you can use, we recommend selecting this option and providing your key pair. Otherwise you can choose to use a password instead.



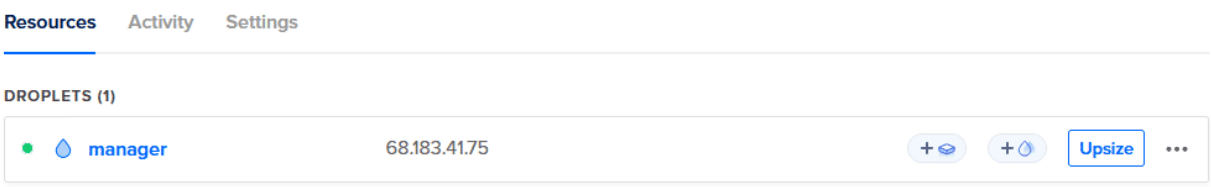
You will need to have SSH key authentication configured to create the Kubernetes cluster machines later on in this course, so we recommend configuring this now.



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



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.



We'll use this IP address to SSH in (using the credentials we provided earlier) and complete the setup.

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:

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/keyrin
chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docke
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") sta
  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:lts   "/portainer"   28 seconds ago
```

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

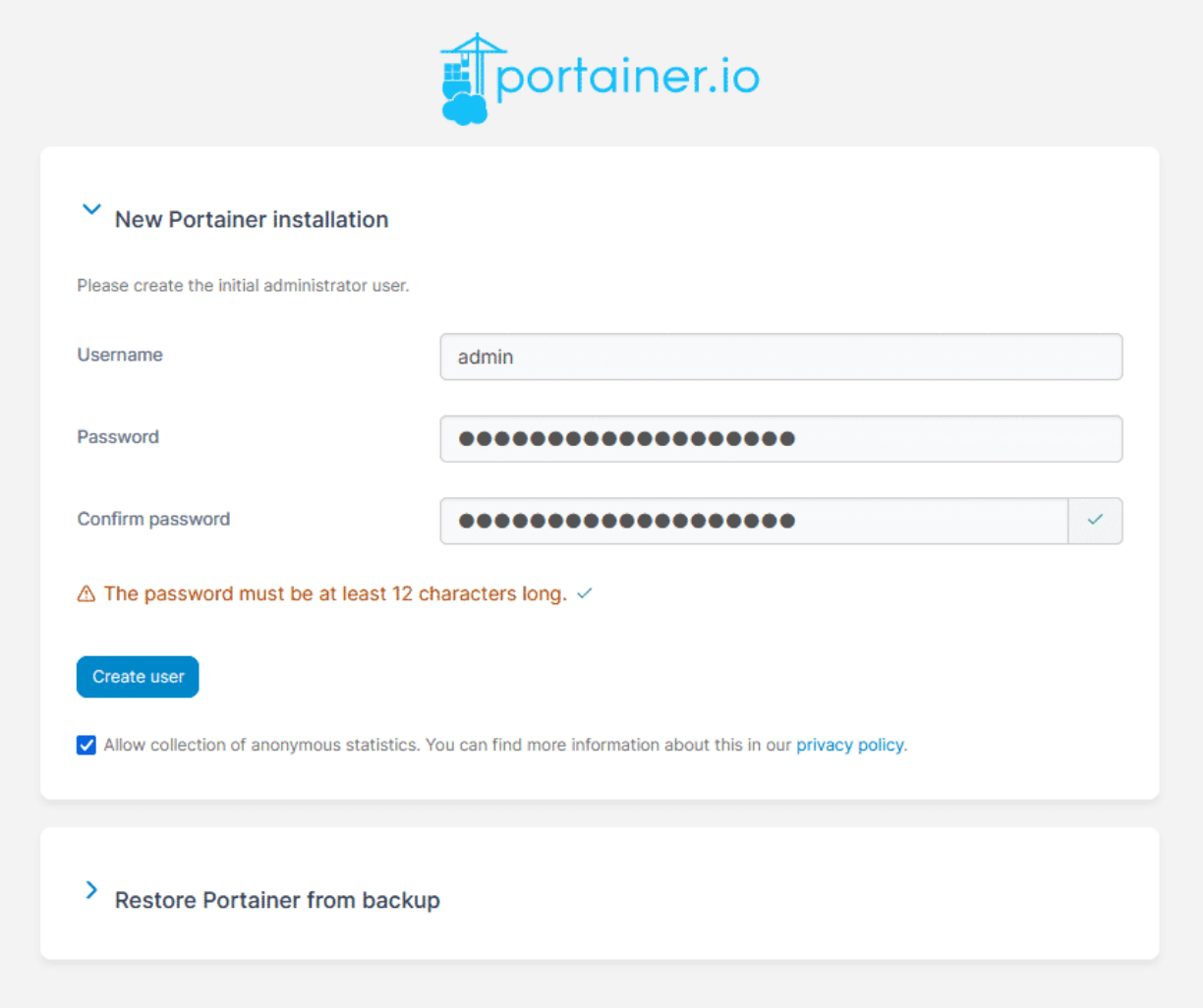
This must be done within 5 minutes of starting the Portainer container for the first time. This is a security measure. If the initial setup has not been completed within 5 minutes of starting, the container will stop listening and will need to be recreated.

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

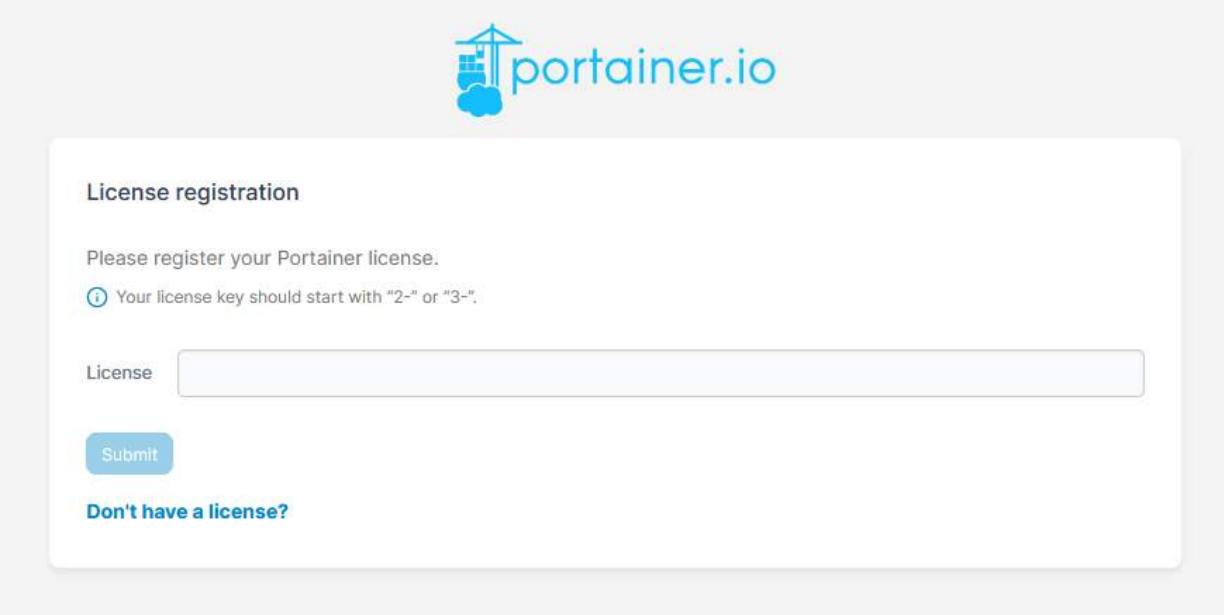
```
https://ipaddress:9443/
```

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

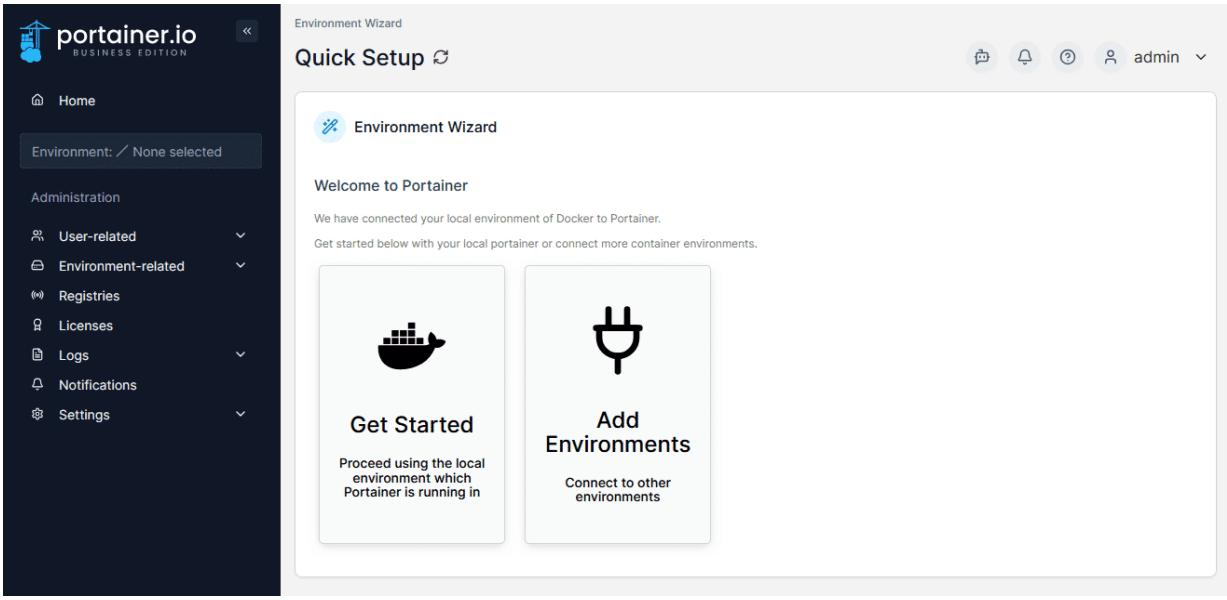
First we must set up an initial administrator user. You can choose to use a different username or stick with the default of `admin`. Enter a secure password, confirm it, and click **Create user**.



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



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.



CONTINUE

4

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.

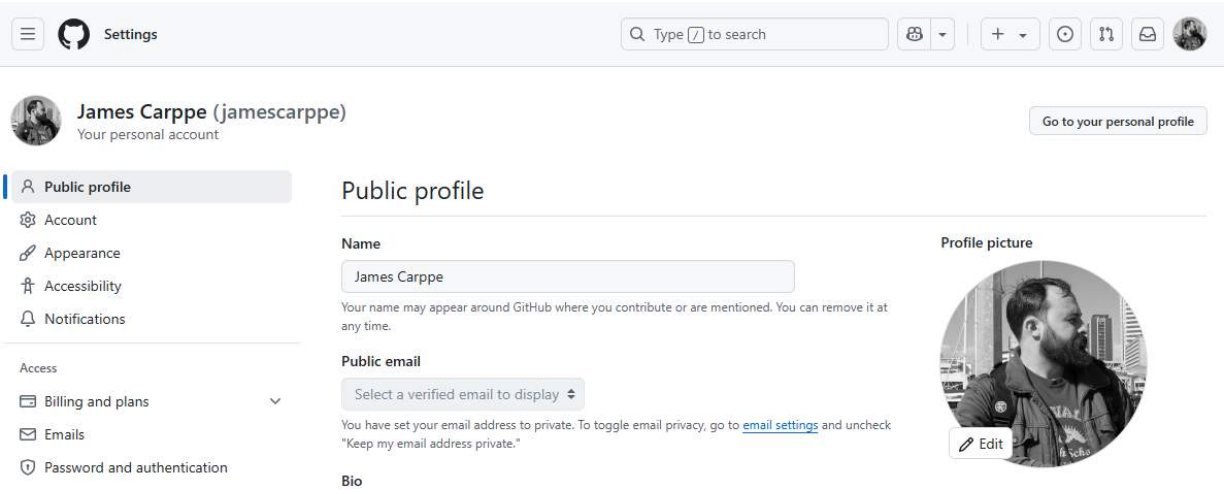
START

1

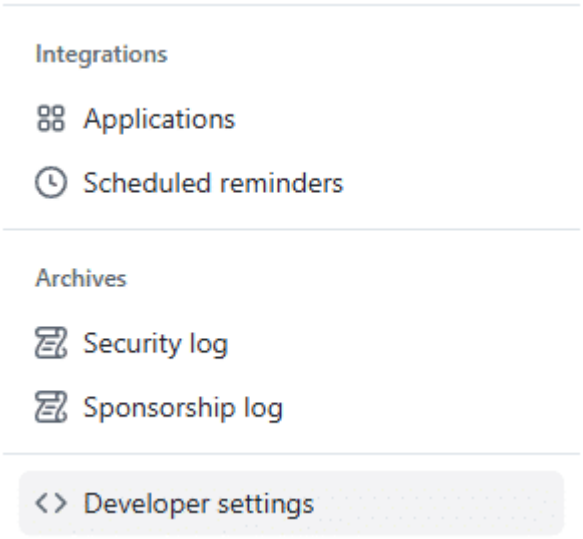
Setting up GitHub as an OAuth provider

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.

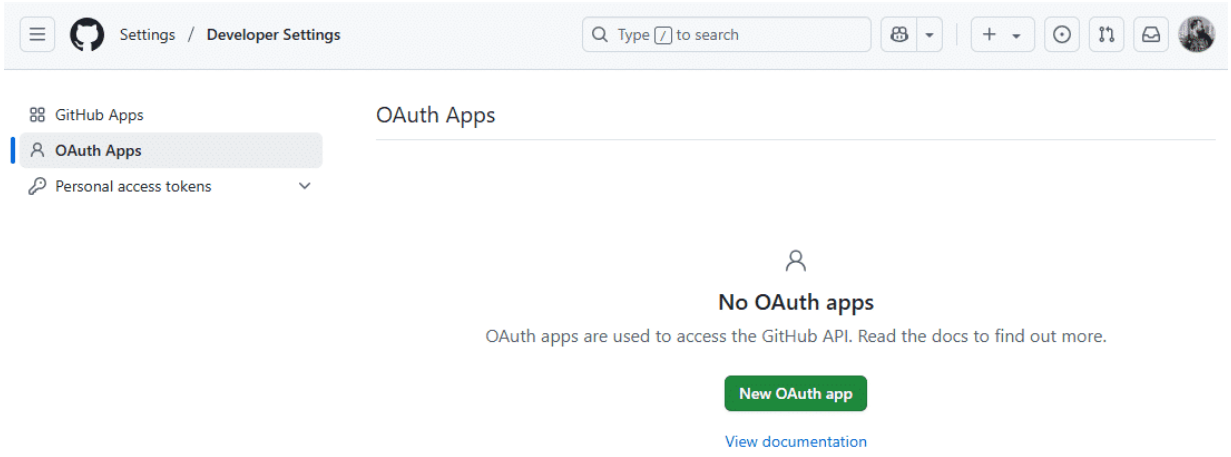
Log into GitHub as your user, then click your profile picture in the top right and select **Settings**. You'll be taken to your profile page.



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



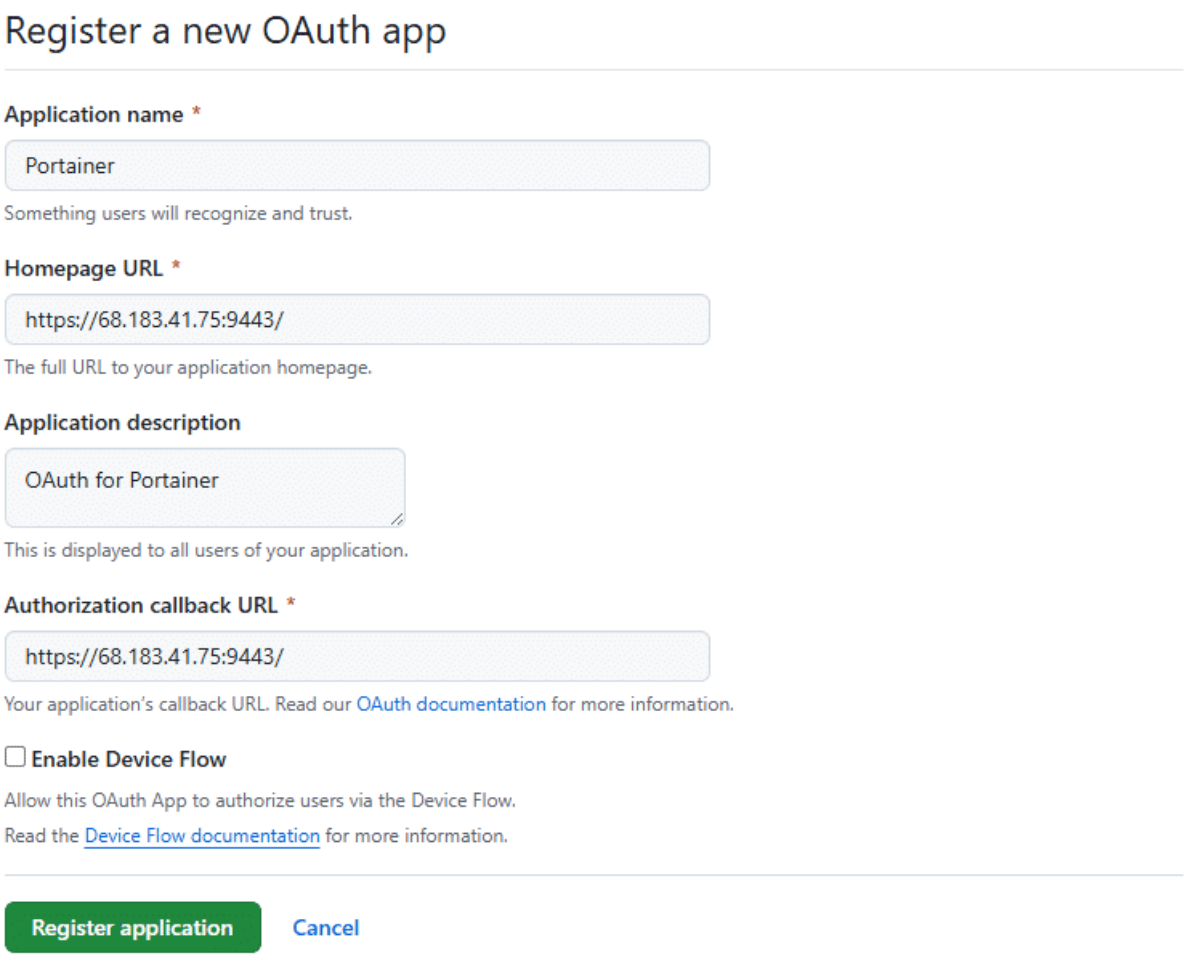
Next, select **OAuth Apps** from the left menu. This will take you to a list of your OAuth apps configured in GitHub. In my case, I have none set up currently.



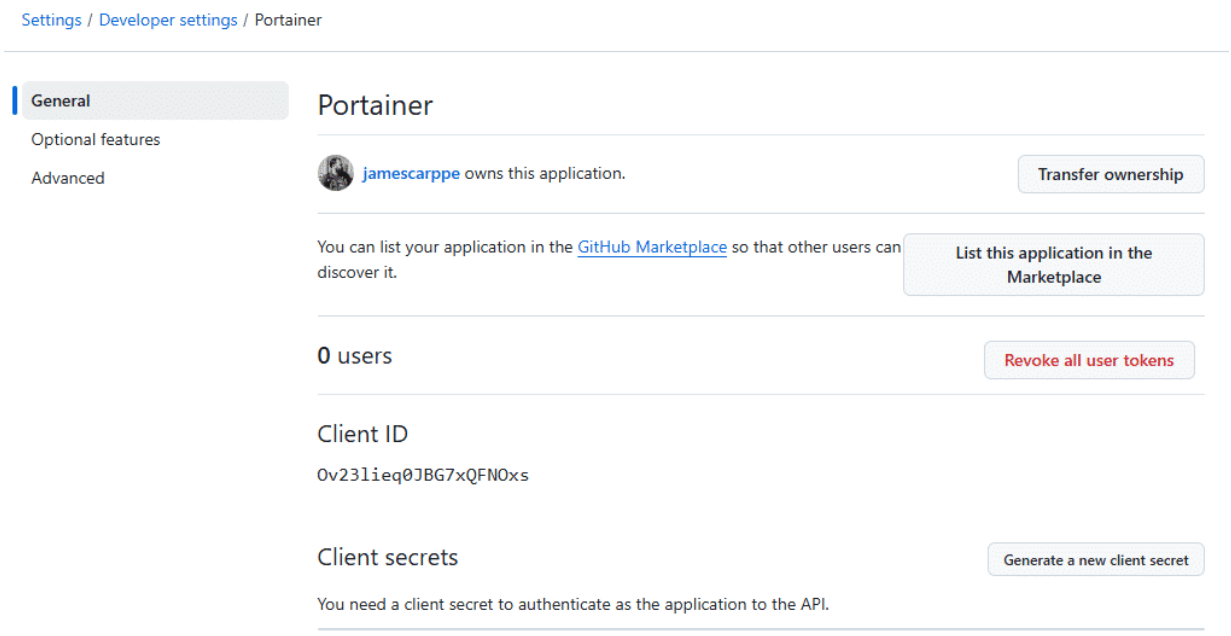
We'll create a new app now for Portainer. Click the **New OAuth app** button then fill out the form.

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.

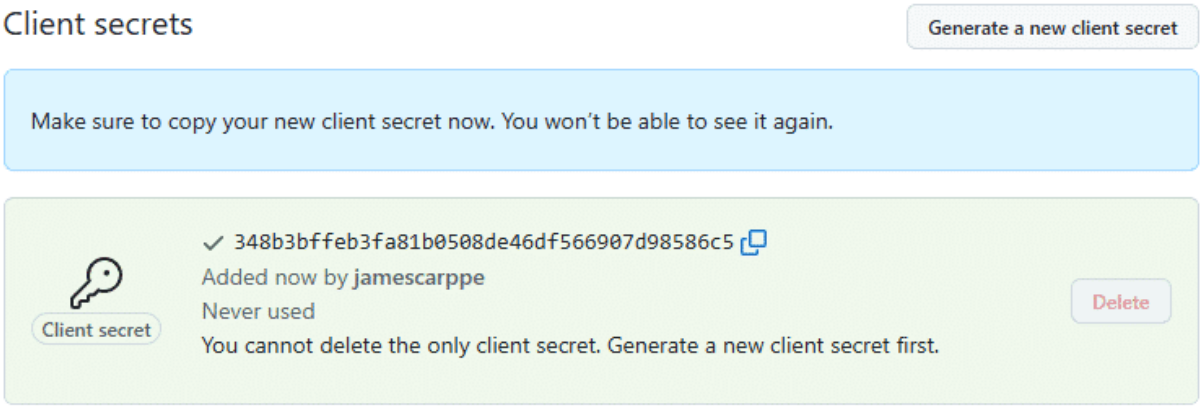


Once you have the form filled out as required, click **Register application**. The application will be created and you'll be taken to the details page. From here, take note of the **Client ID** value displayed - we'll need this for the next step.



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.



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

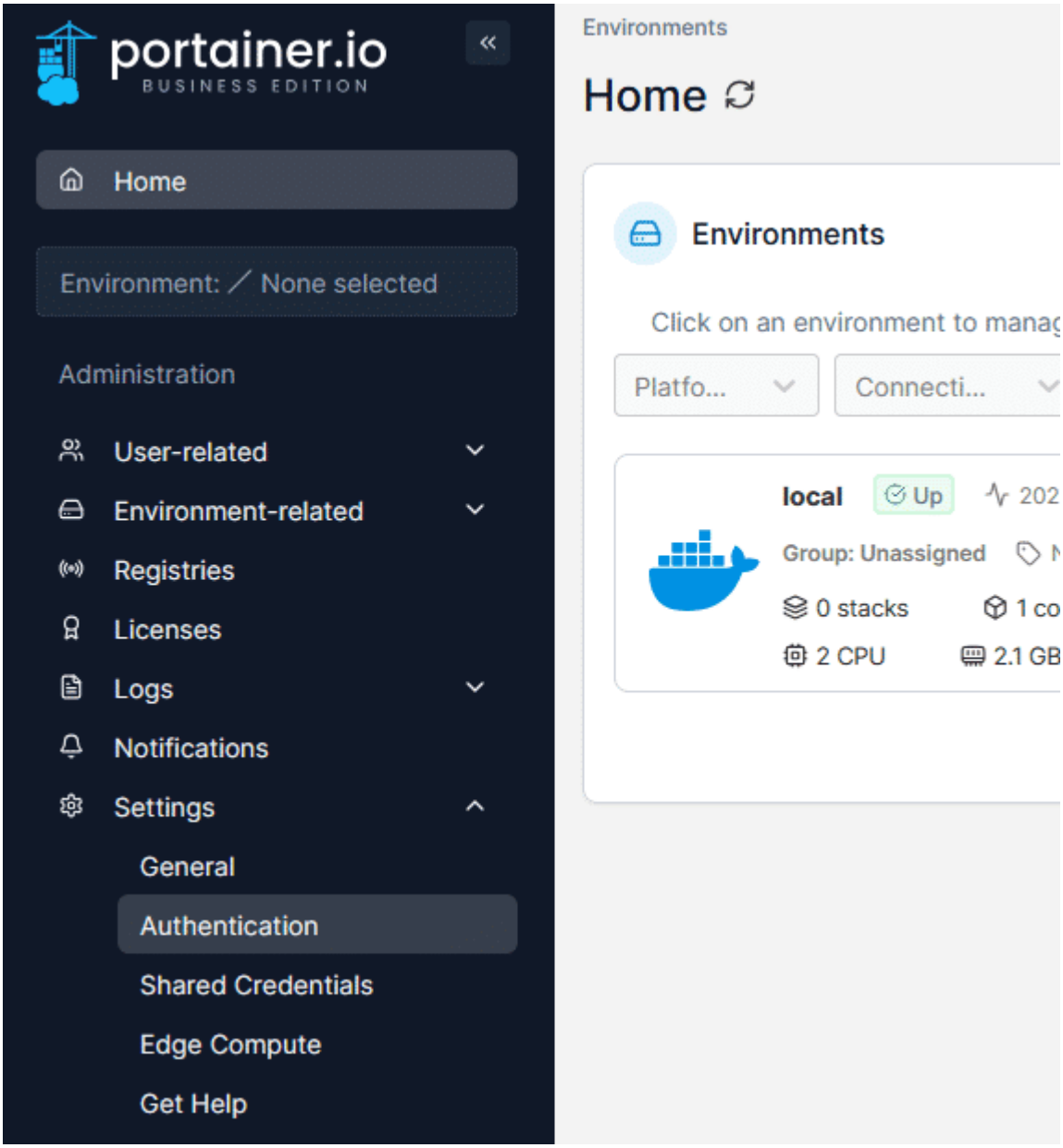
CONTINUE

2

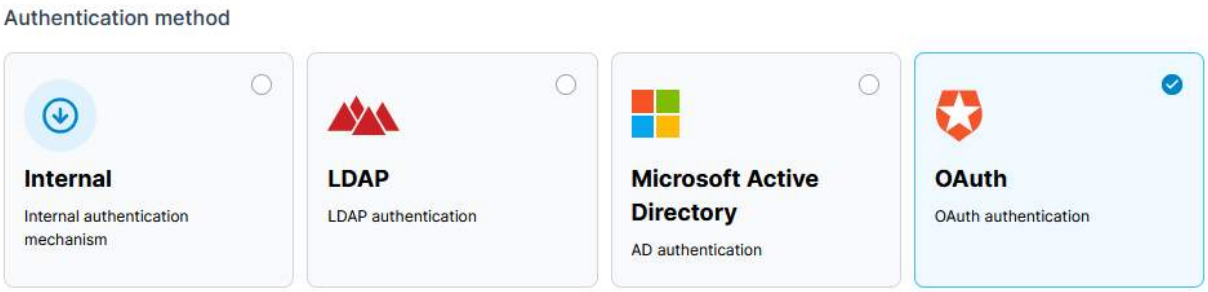
Configuring Portainer for OAuth

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

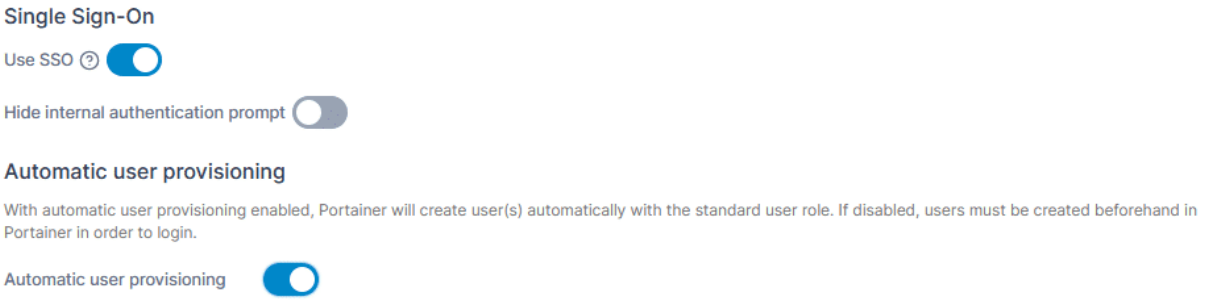
In Portainer, scroll down and expand the **Settings** menu at the bottom left and select **Authentication**.



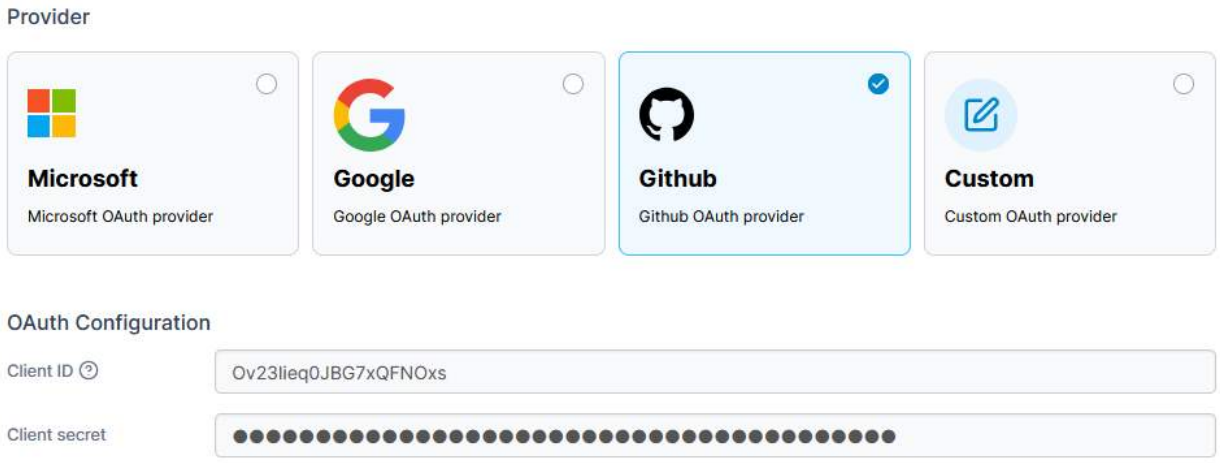
Here you'll see the authentication options for your Portainer instance. The **Internal** method is currently selected, but we want to change this so select **OAuth** instead.



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.



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.



With this completed, click the **Save settings** button to apply your configuration changes.

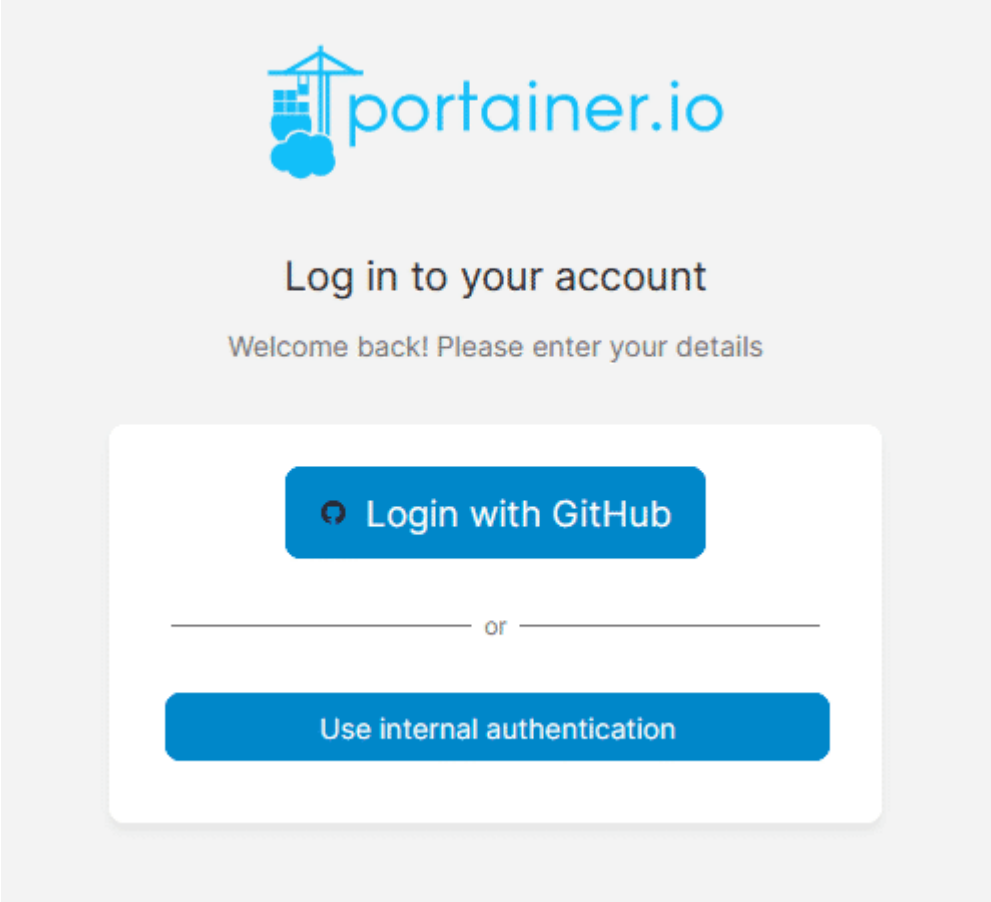
CONTINUE

3

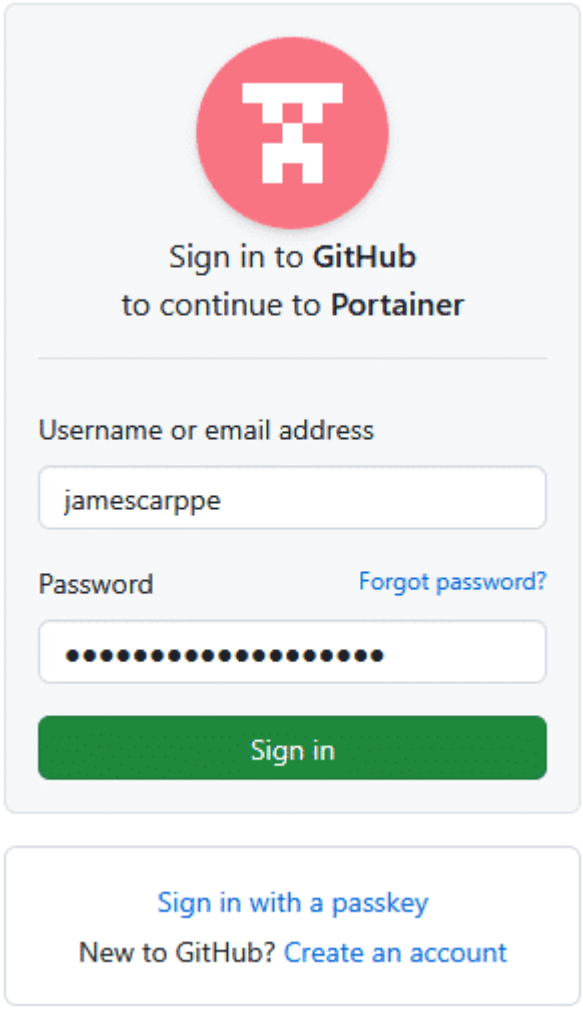
Testing OAuth

Let's now test to make sure that the OAuth configuration was applied successfully. Log out of Portainer by clicking your username in the top right and selecting **Log out**.

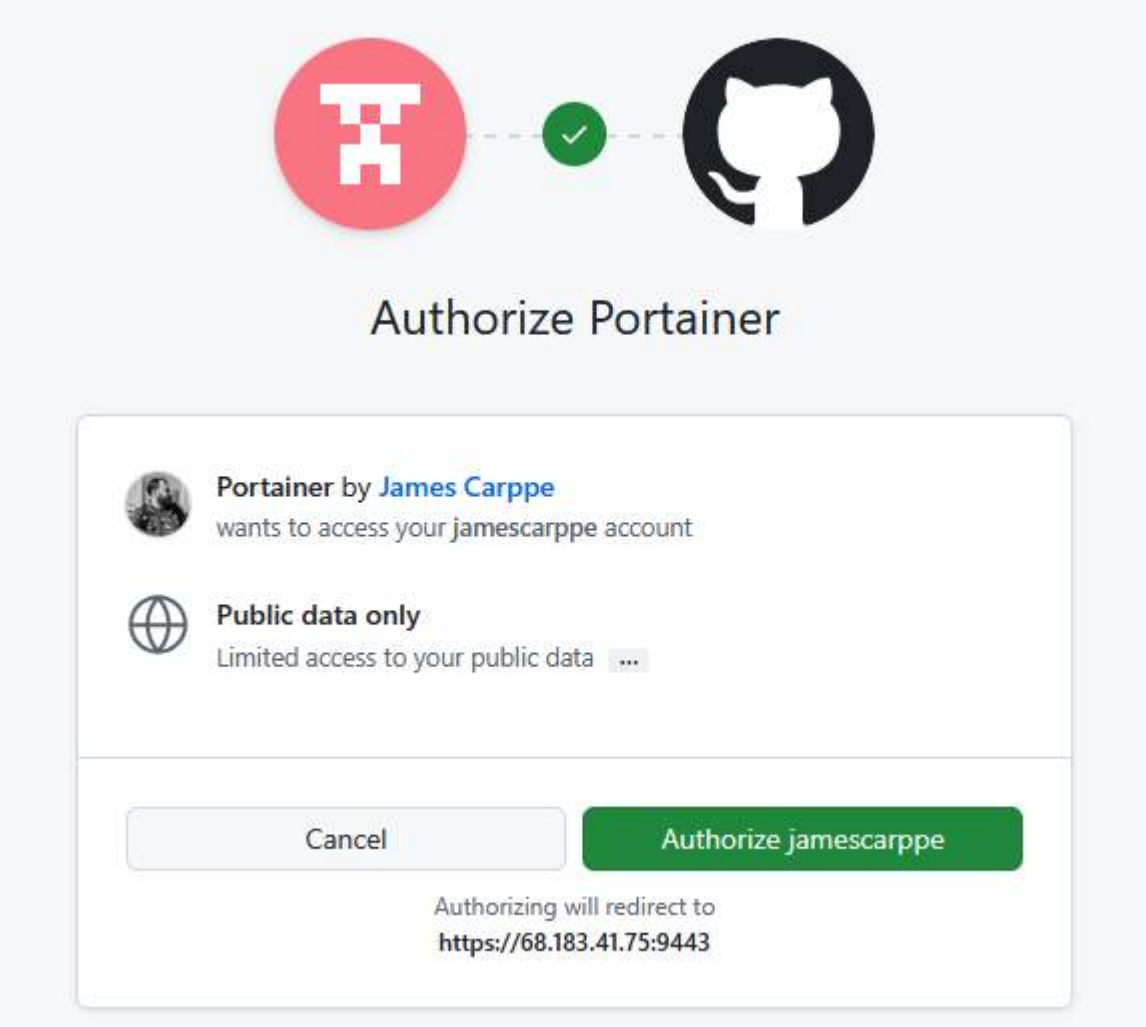
Once you're logged out, return to the URL for your Portainer instance. The login screen should look a little bit different now.



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

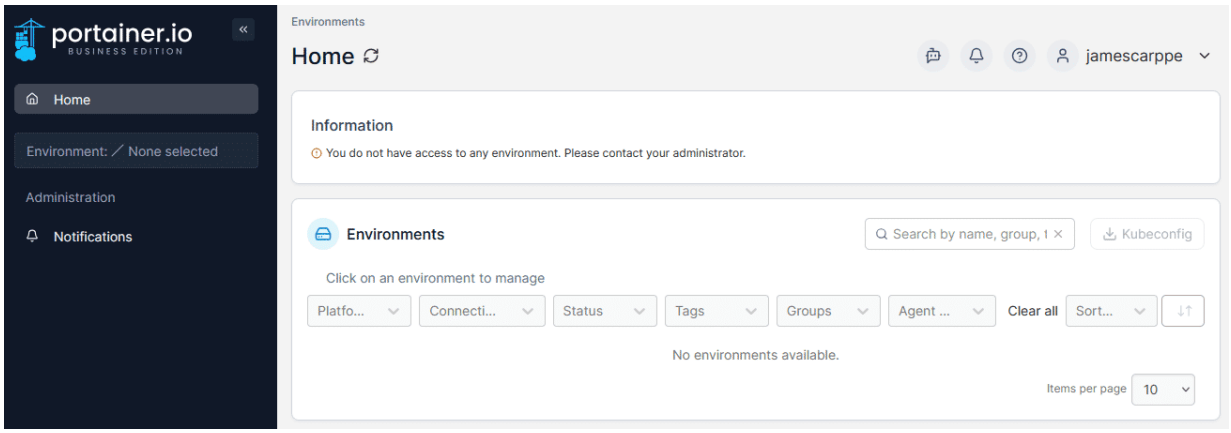


As this is the first time you're logging in to this OAuth application, you will be asked to authorize access to your GitHub account from your OAuth application. Click the **Authorize** button to proceed.



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.

Lesson 4 of 11

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:

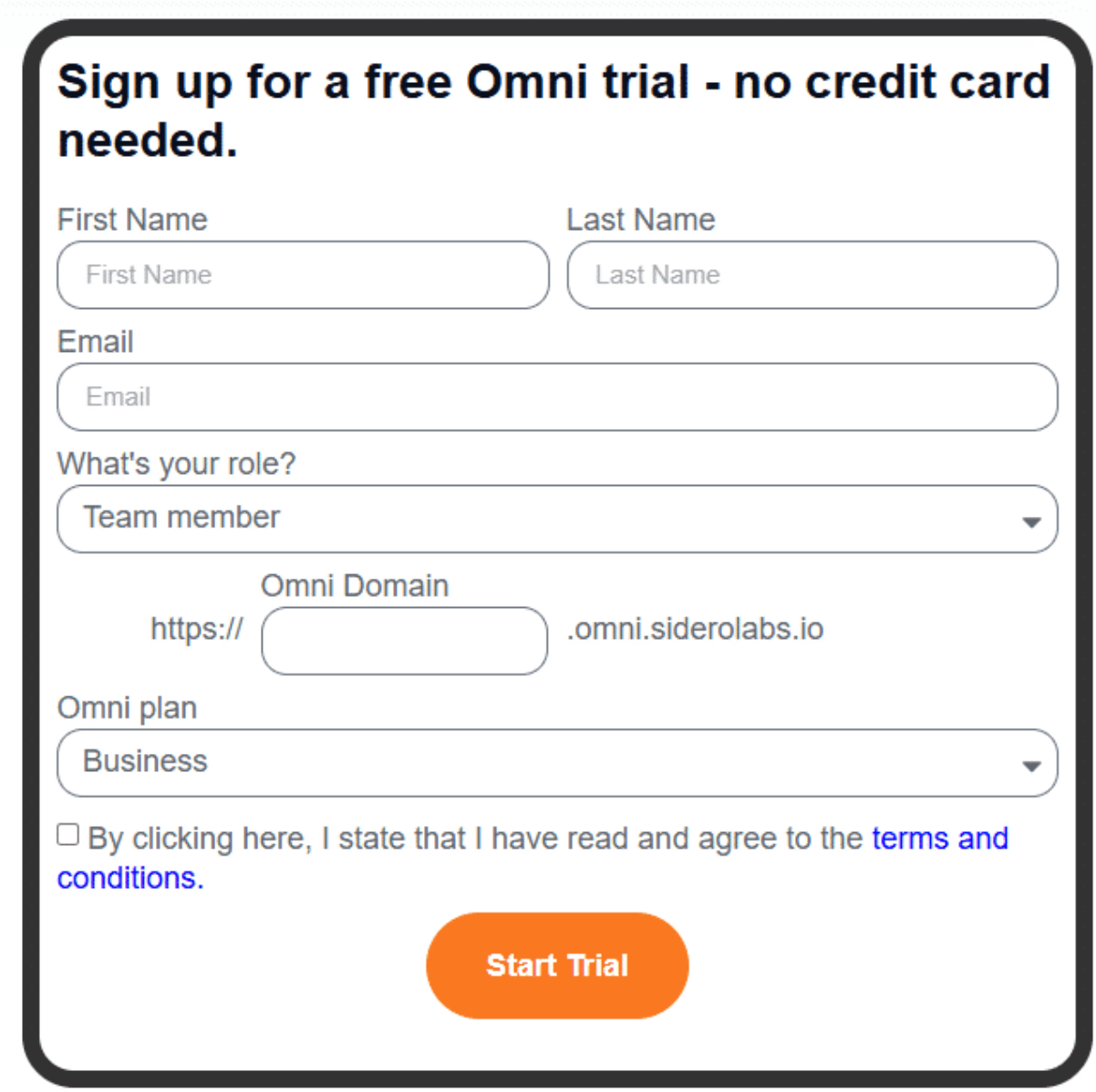
1. Sign up for an Omni account with Sidero.
2. Create a Service Account within our Omni account.
3. Download the Digital Ocean machine image we will use to provision our cluster machines.
4. Add the Omni Service Account to our Portainer installation.

START

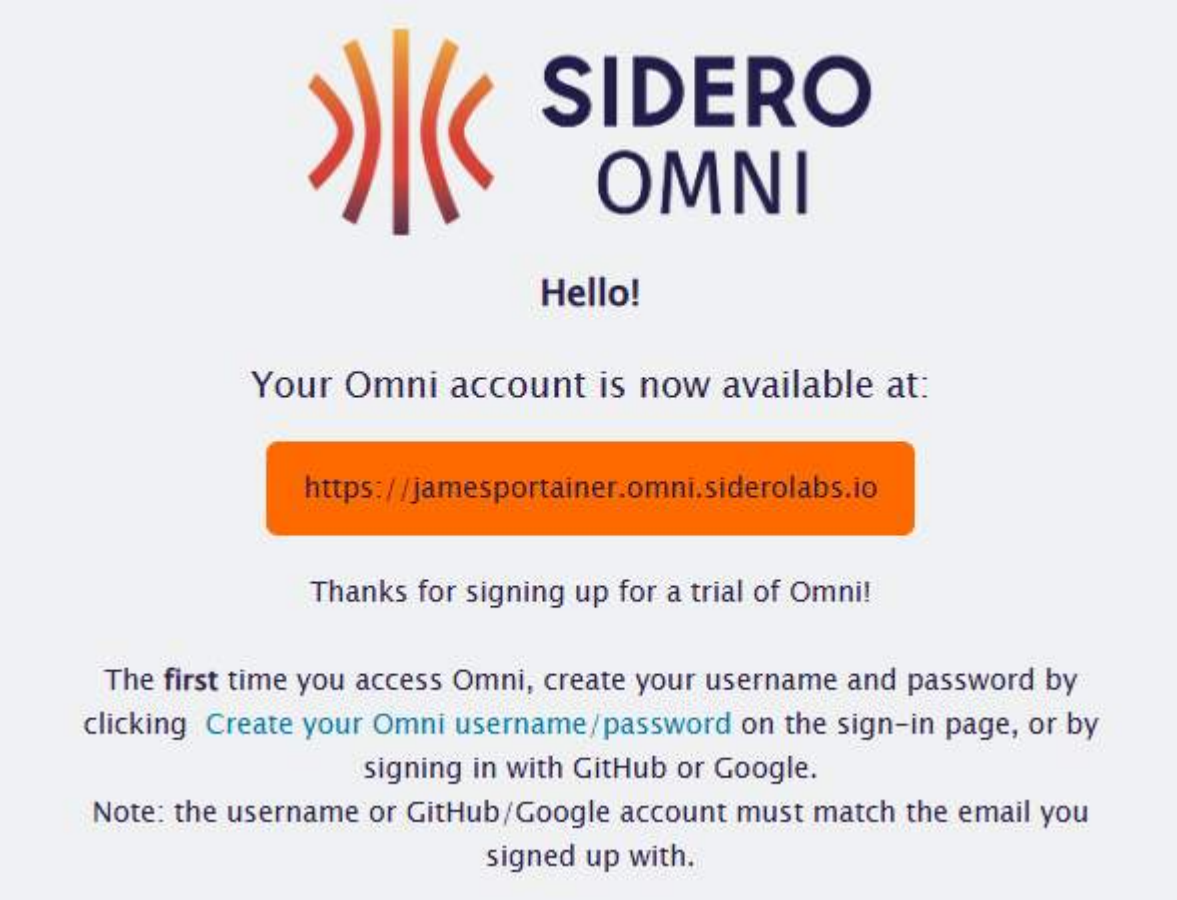
1

Creating an Omni account

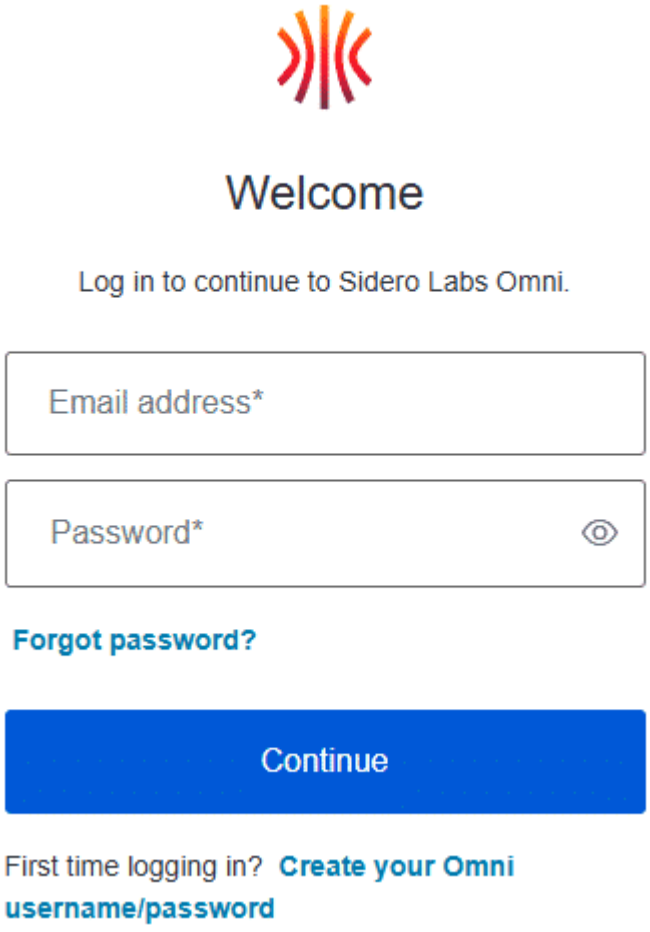
Sidero offer a free trial of Omni SaaS that you can sign up for on their website. Head there and fill out the form, then click **Start Trial**.



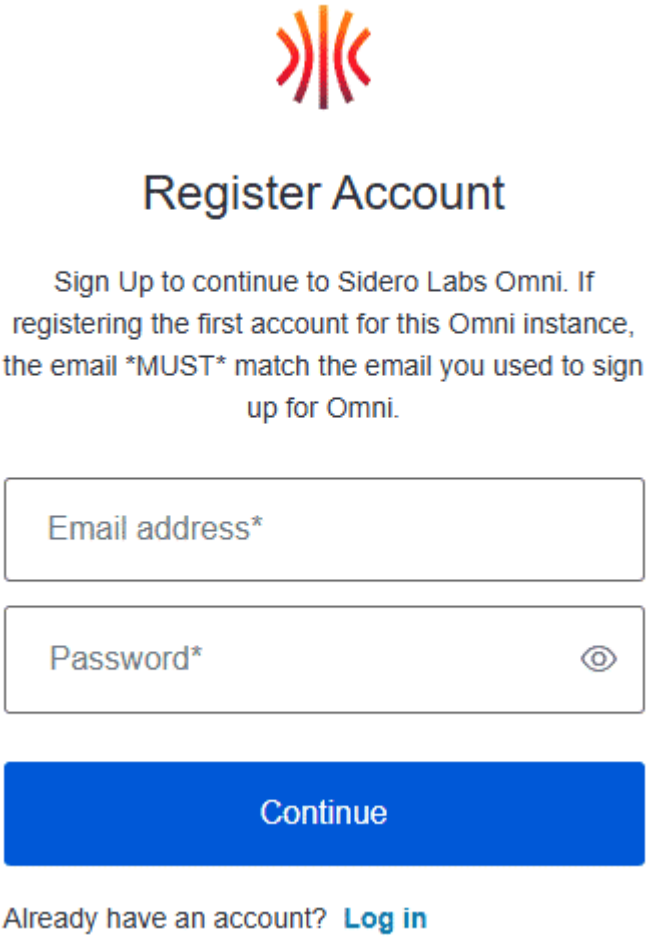
Once you've completed the signup you'll receive an email that will include your Omni account address.



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



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



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!

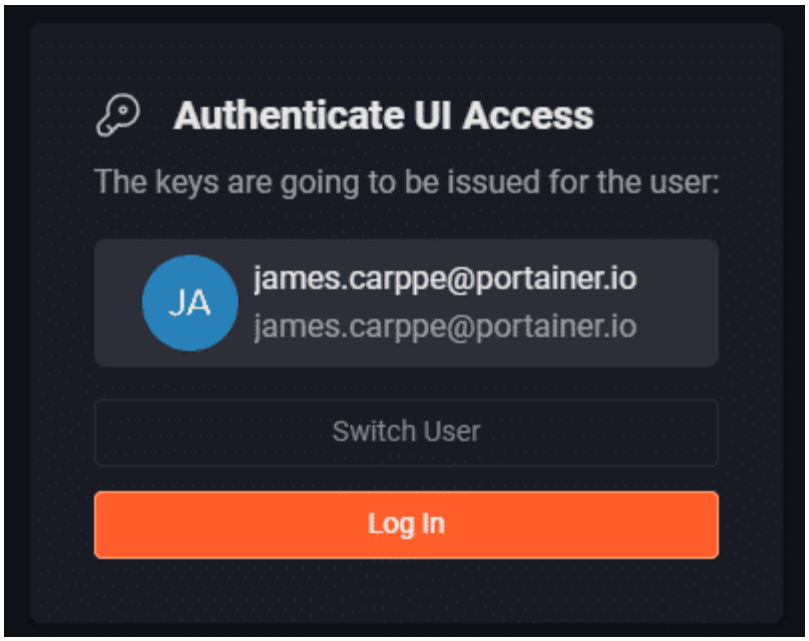
Thank you for signing up. Please verify your email address by clicking the following link:

Confirm my account

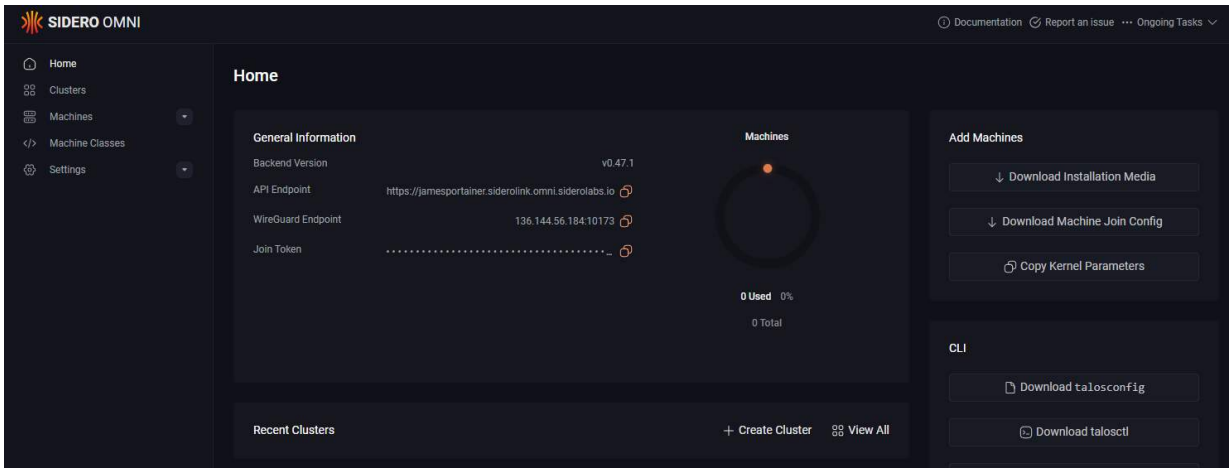
If you are having any issues with your account, please don't hesitate to contact us by replying to this mail.

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.



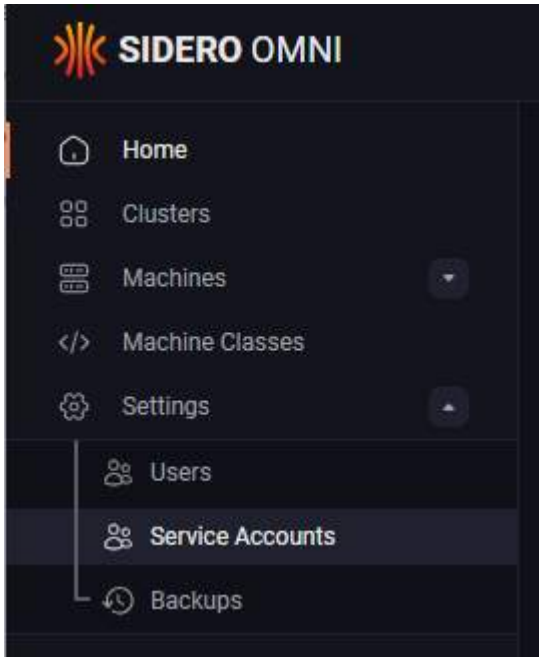
CONTINUE

2

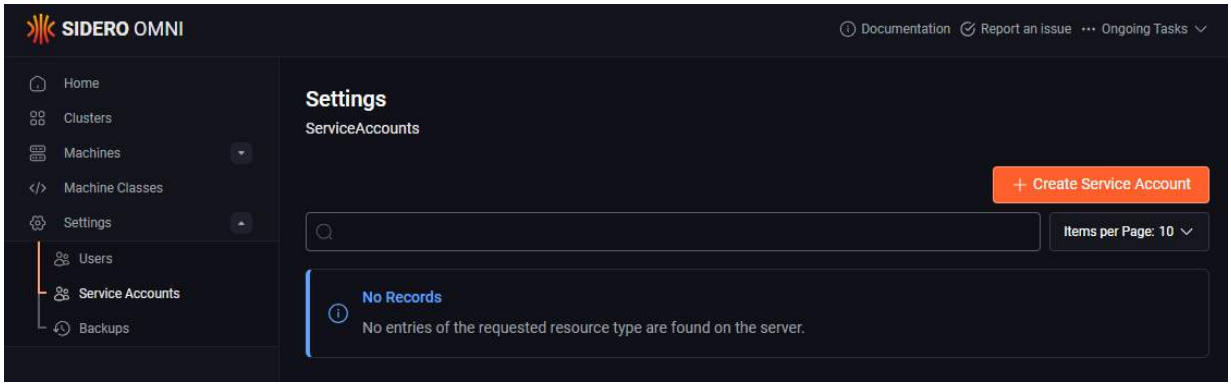
Creating an Omni Service Account

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.

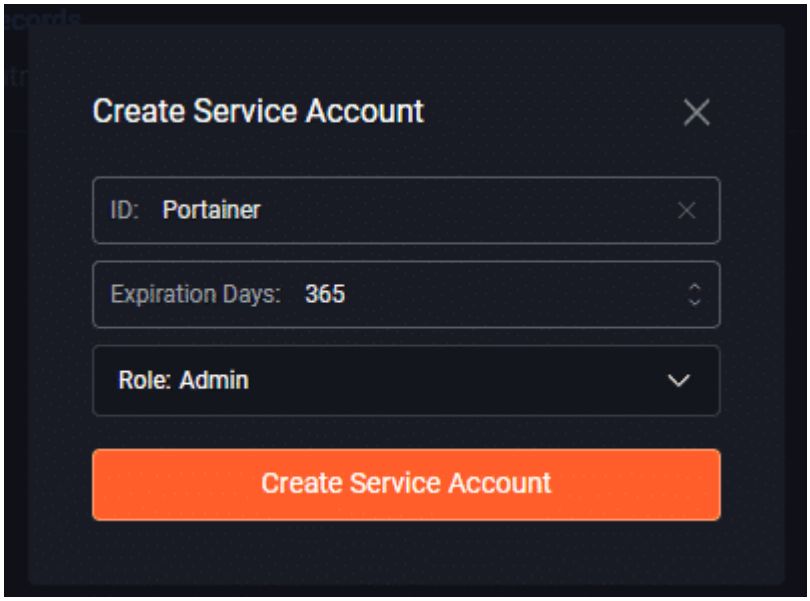
From the Omni dashboard, expand the **Settings** menu on the left and click **Service Accounts**.



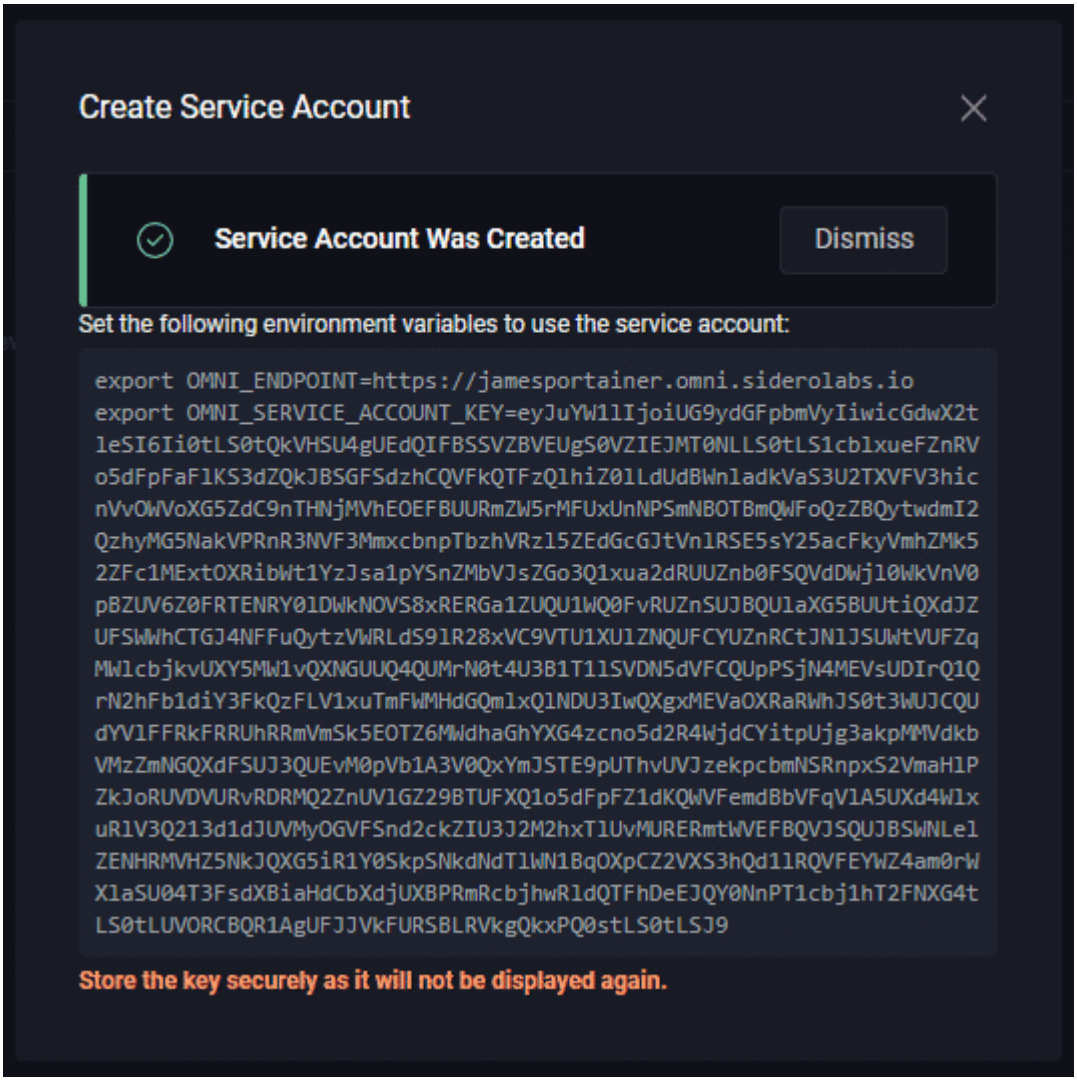
On the **Service Accounts** page we'll see that we have none currently created. So let's create one. Click the **Create Service Account** button.



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**.



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.



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

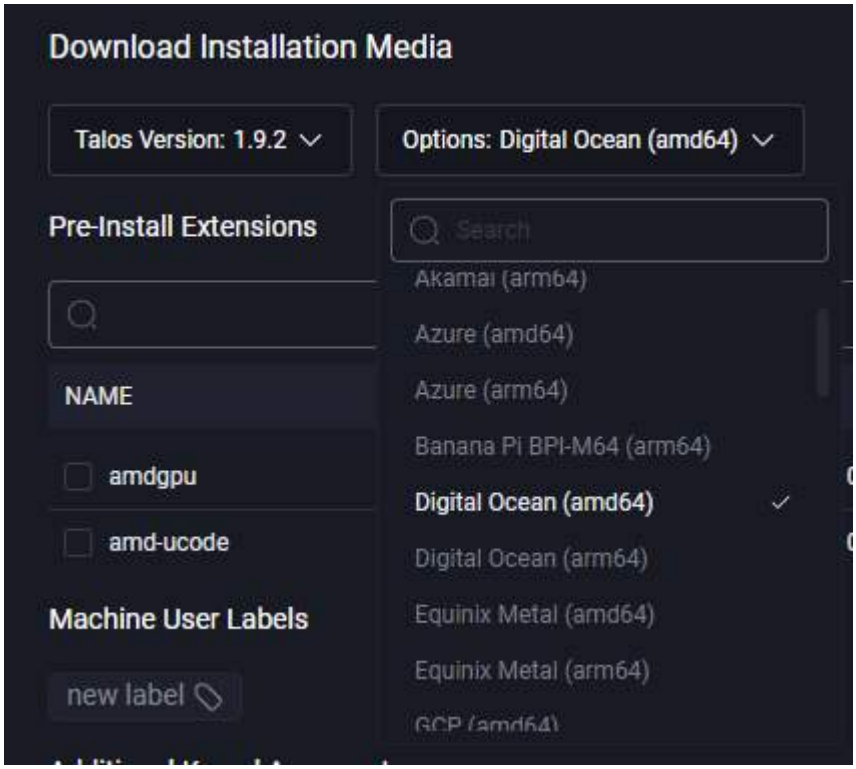
3

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).**



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.

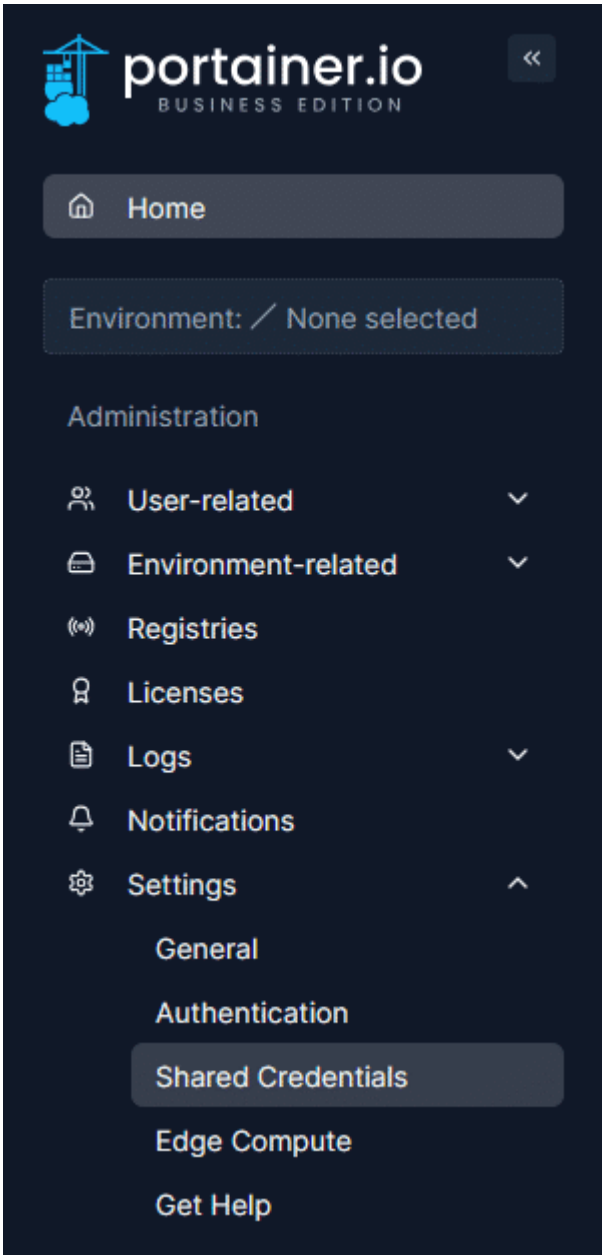
We're now done in the Omni dashboard. Next we'll add the Omni Service Account to Portainer.

CONTINUE

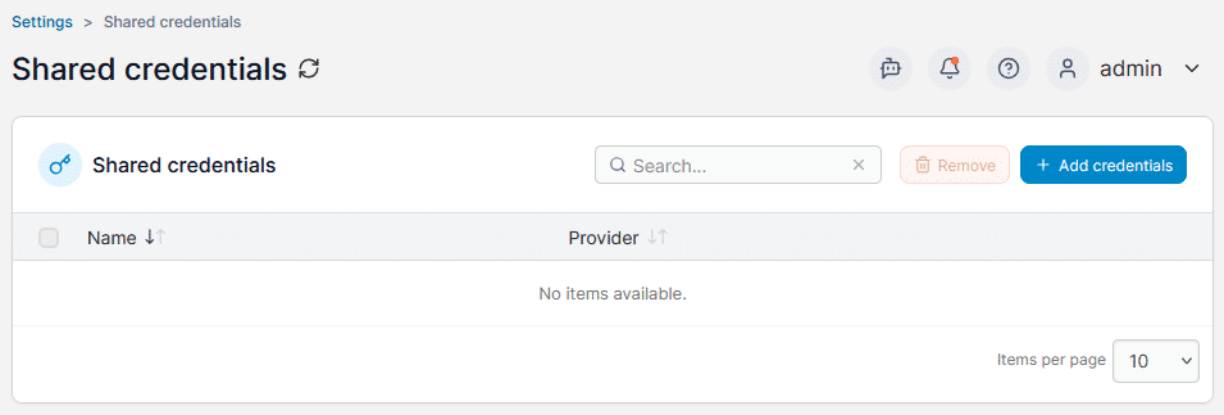
4

Adding the Omni Service Account to Portainer

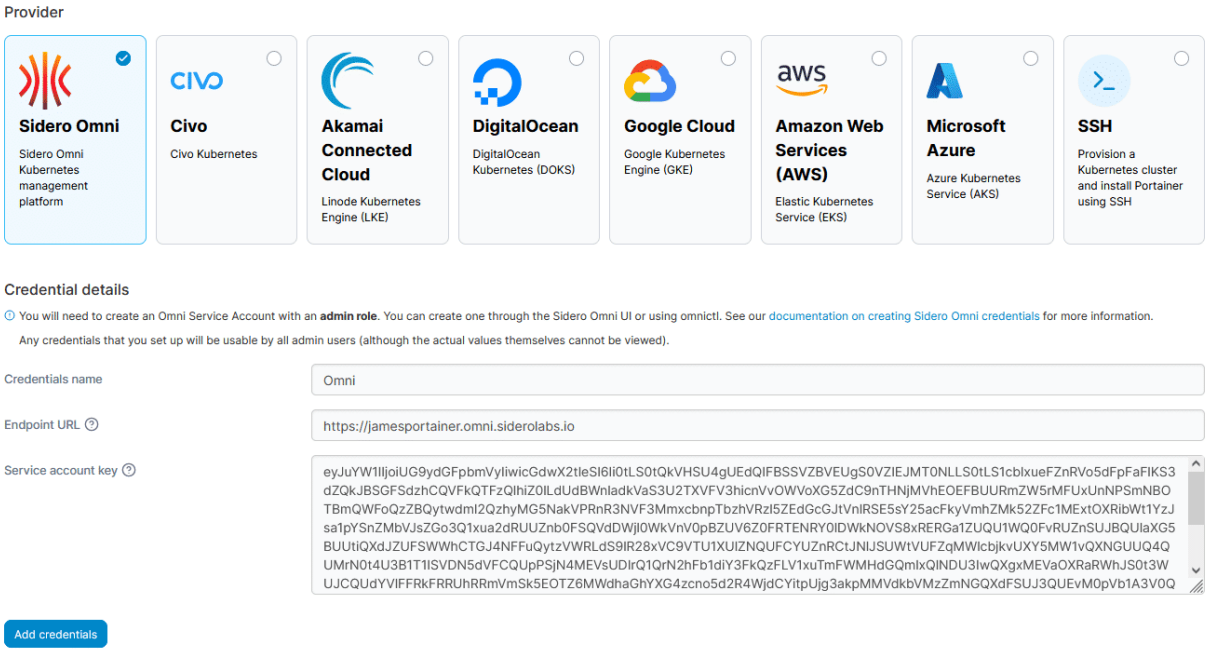
Log into your Portainer server as the administrator. Expand the **Settings** menu on the left and click **Shared Credentials**.



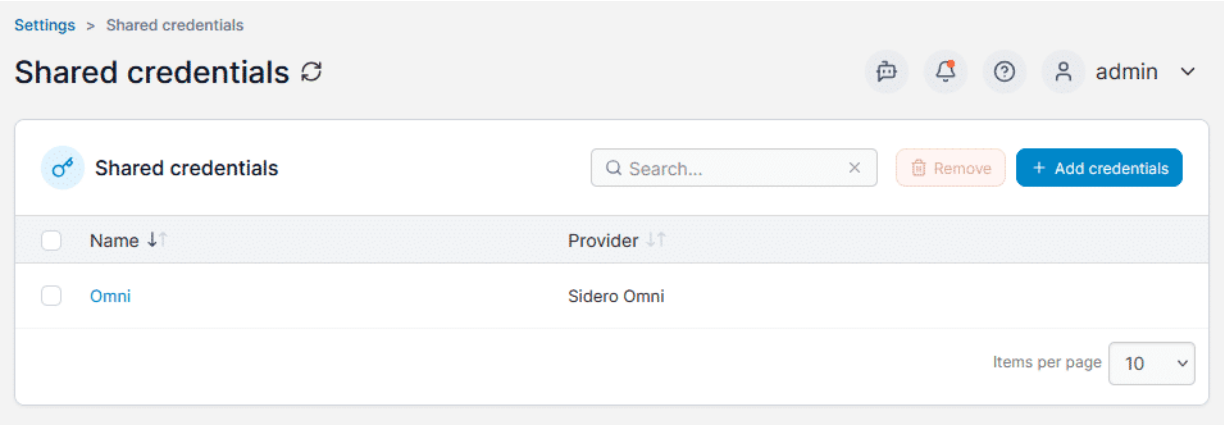
On this fresh install of Portainer you won't have any credential sets configured. Click the **Add credentials** button to create one.



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**.



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.

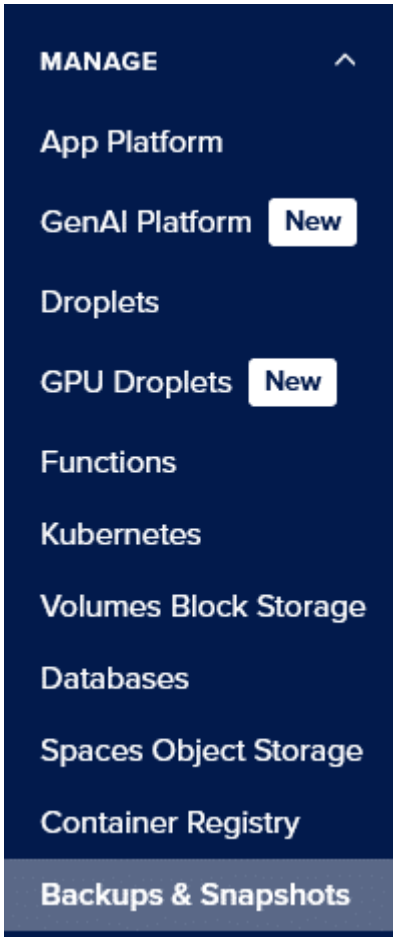
START

1

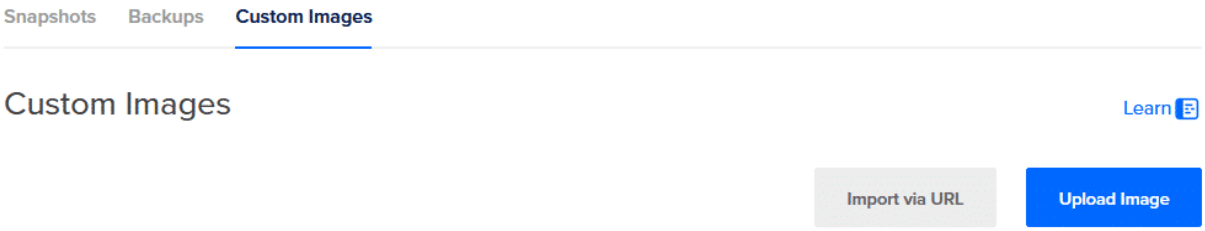
Uploading the machine image

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**.



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

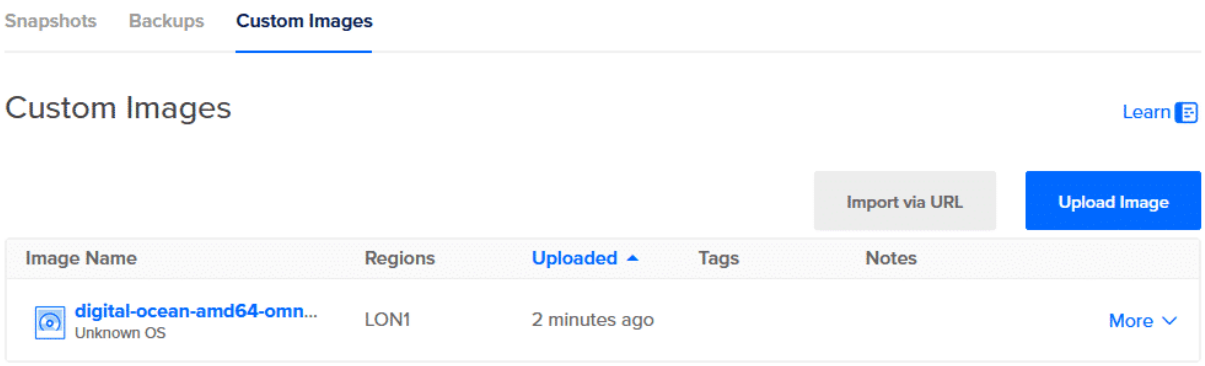
Unknown

Choose a datacenter region

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

New York	San Francisco	Amsterdam
1 2 3	1 2 3	2 3
Singapore	London	Frankfurt
1	1	1
Toronto	Bangalore	Sydney
1	1	1

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.



Next we'll provision some servers using this image.

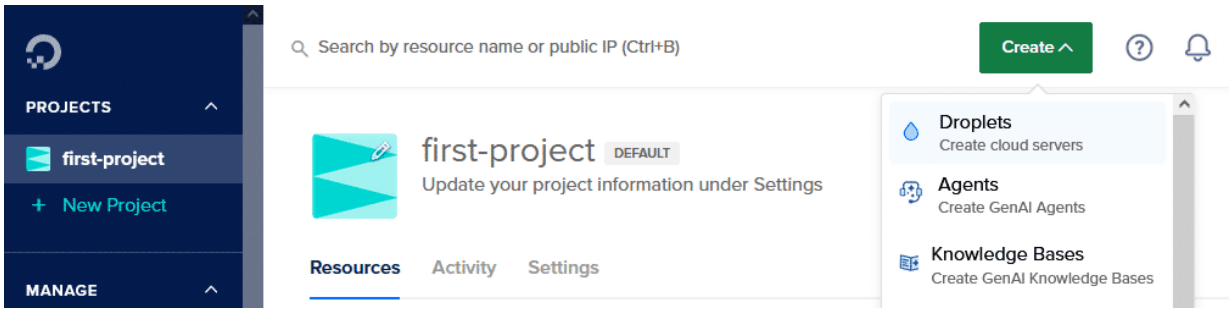
CONTINUE

2

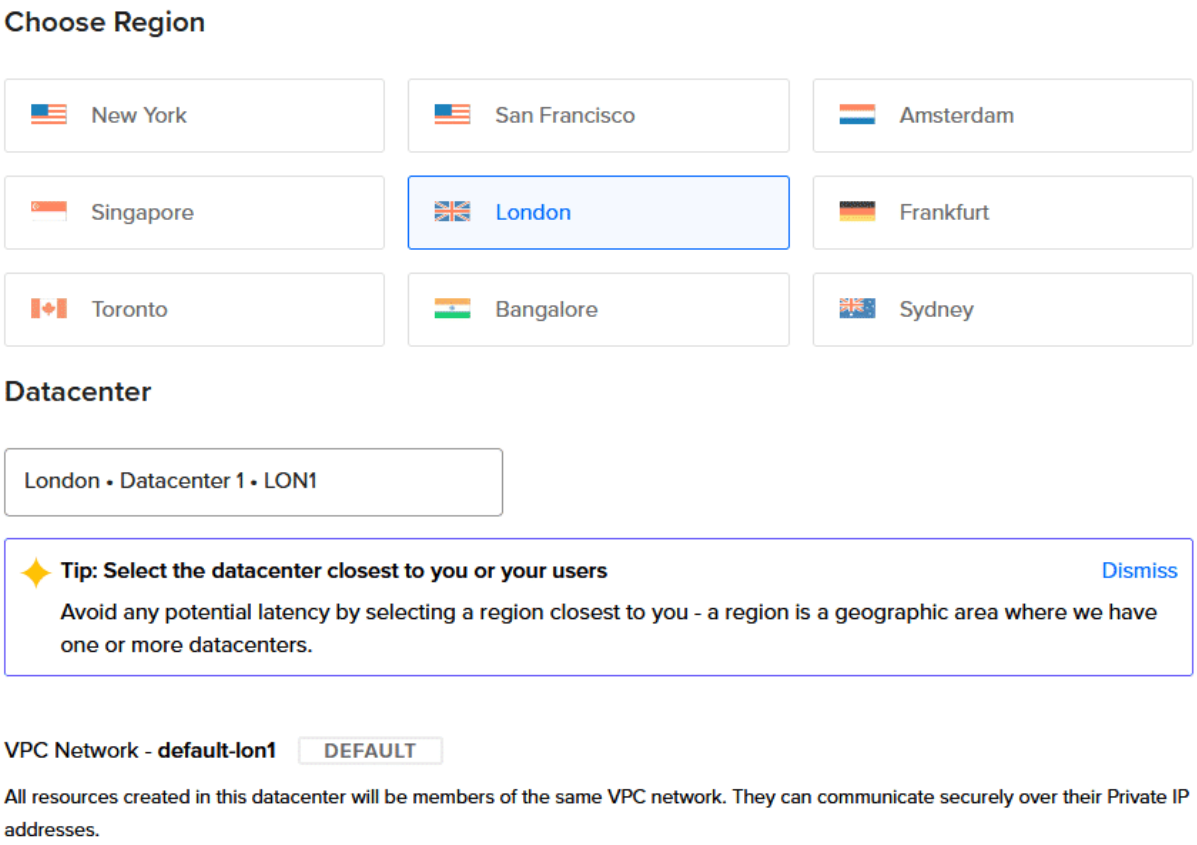
Creating the Talos machines

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.

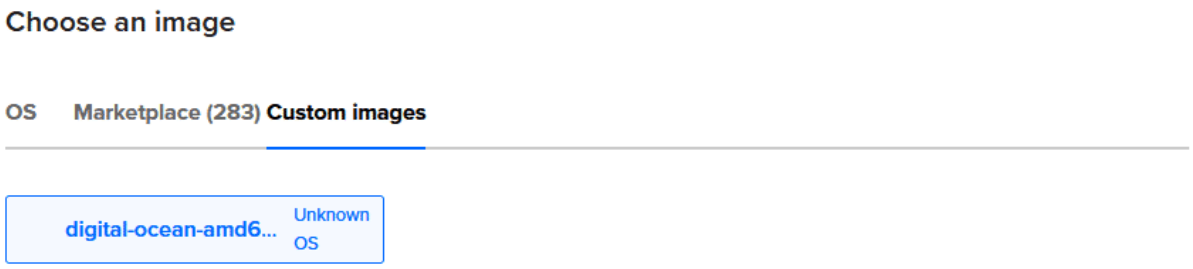
In Digital Ocean, click the **Create** button and choose **Droplets**.



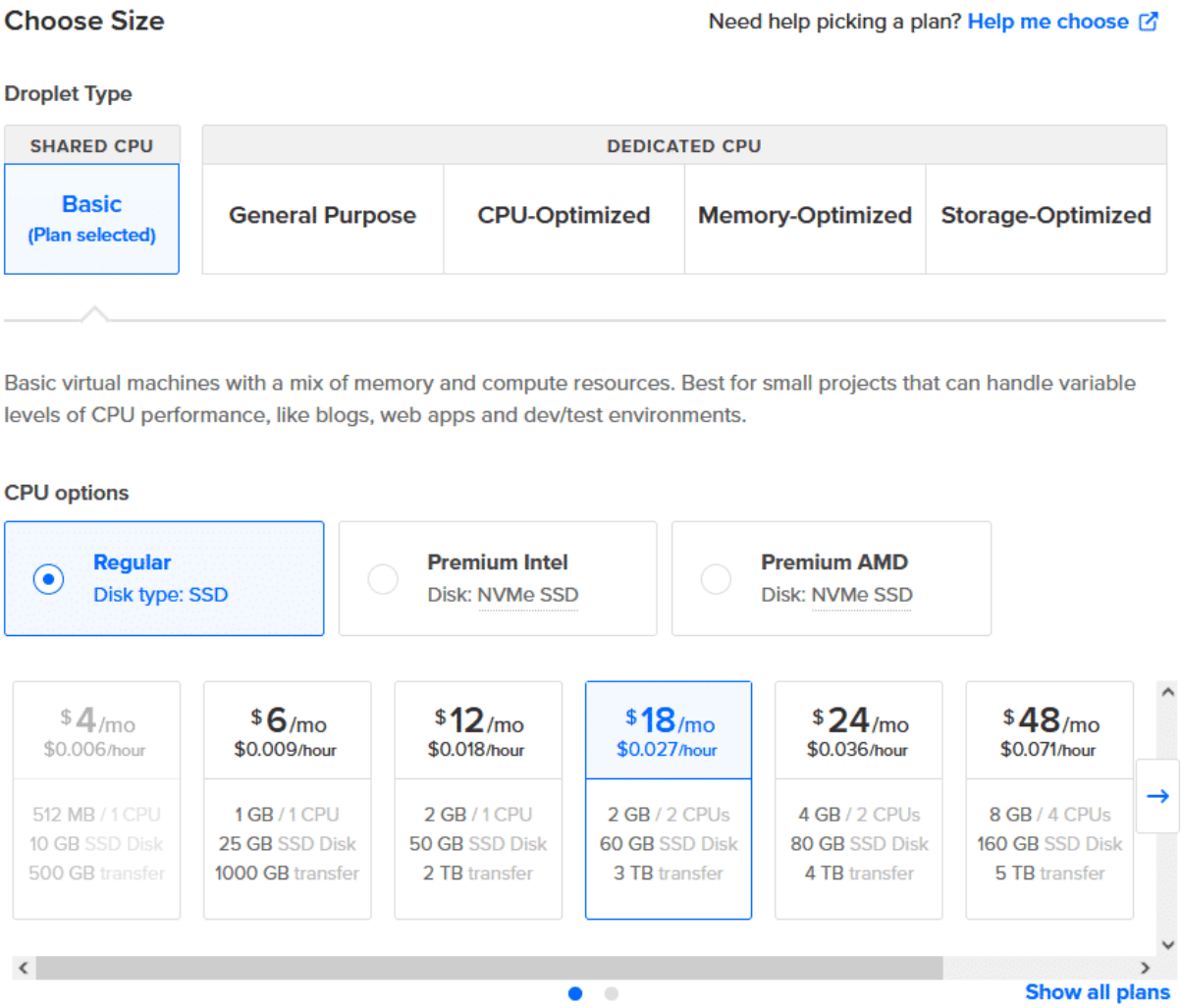
Choose the **region** and **datacenter** you uploaded the custom image to.



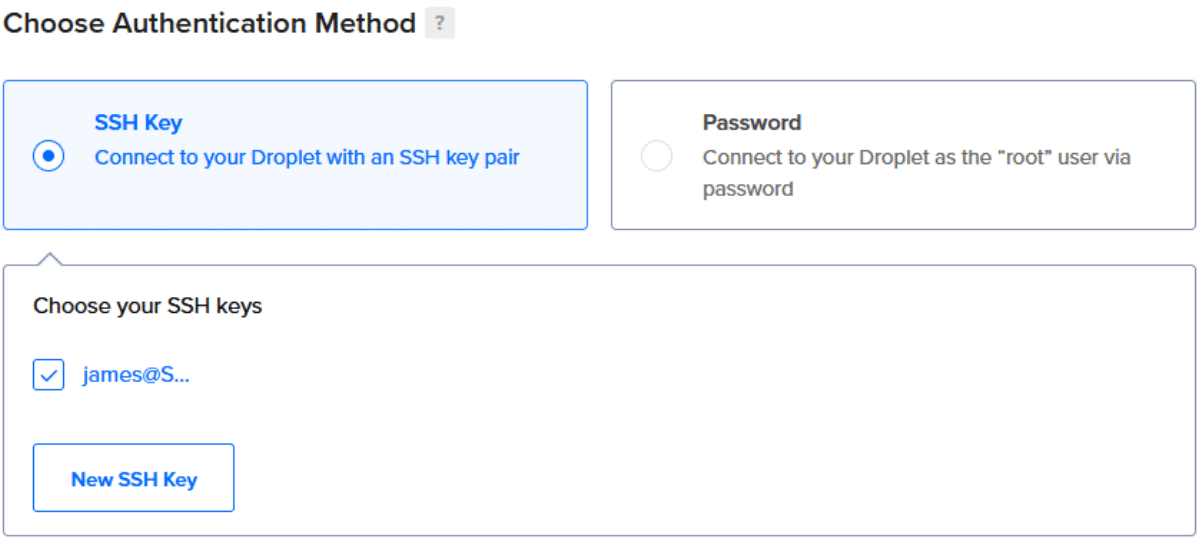
Scroll down to **Choose an image** and click the **Custom images** tab. You should see your custom image listed - select it.



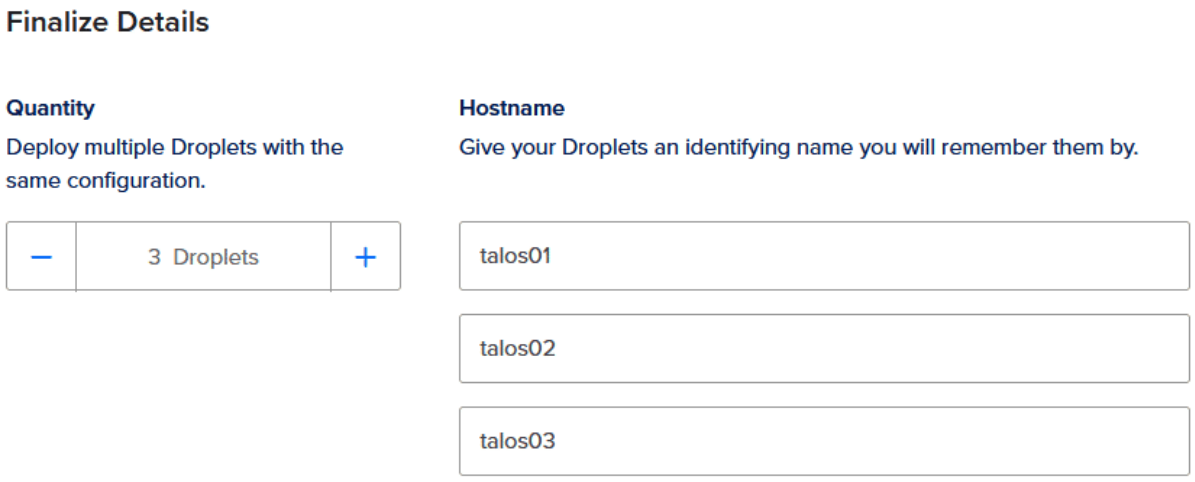
For the droplet size we'll again stick with the `Basic` shared CPU, the `Regular` disk and the 2GB / 2 CPU option.



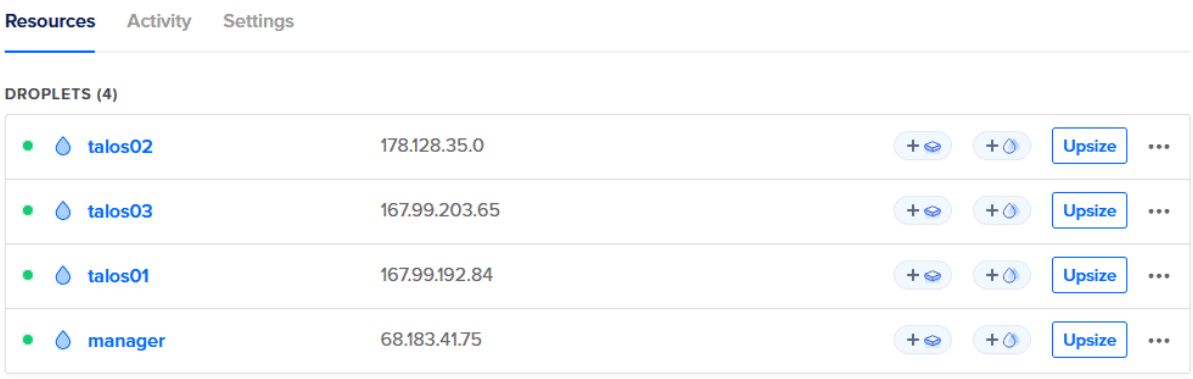
The Talos images require using a **SSH key** for authentication, so if you don't already have one set up from earlier you should do so now. If you already have one, select that.



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

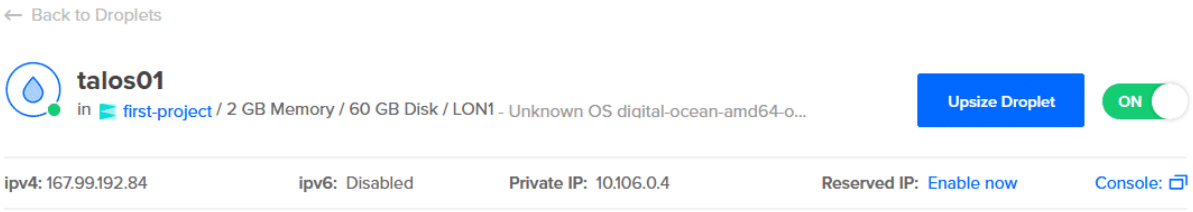


When you're ready, click **Create Droplet**. The servers will begin provisioning, and once complete will appear in your list of droplets.



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`.



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!

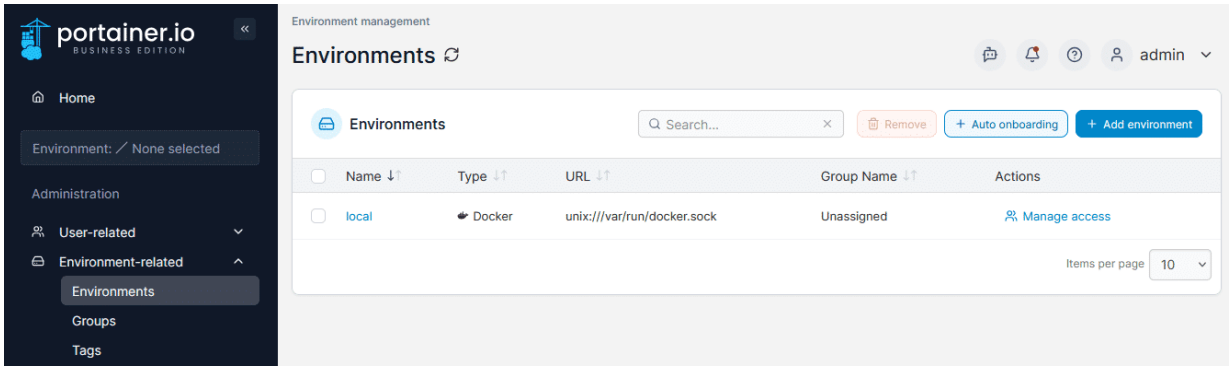
In this lesson we will provision our environment from Portainer on our Digital Ocean machines, with:

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

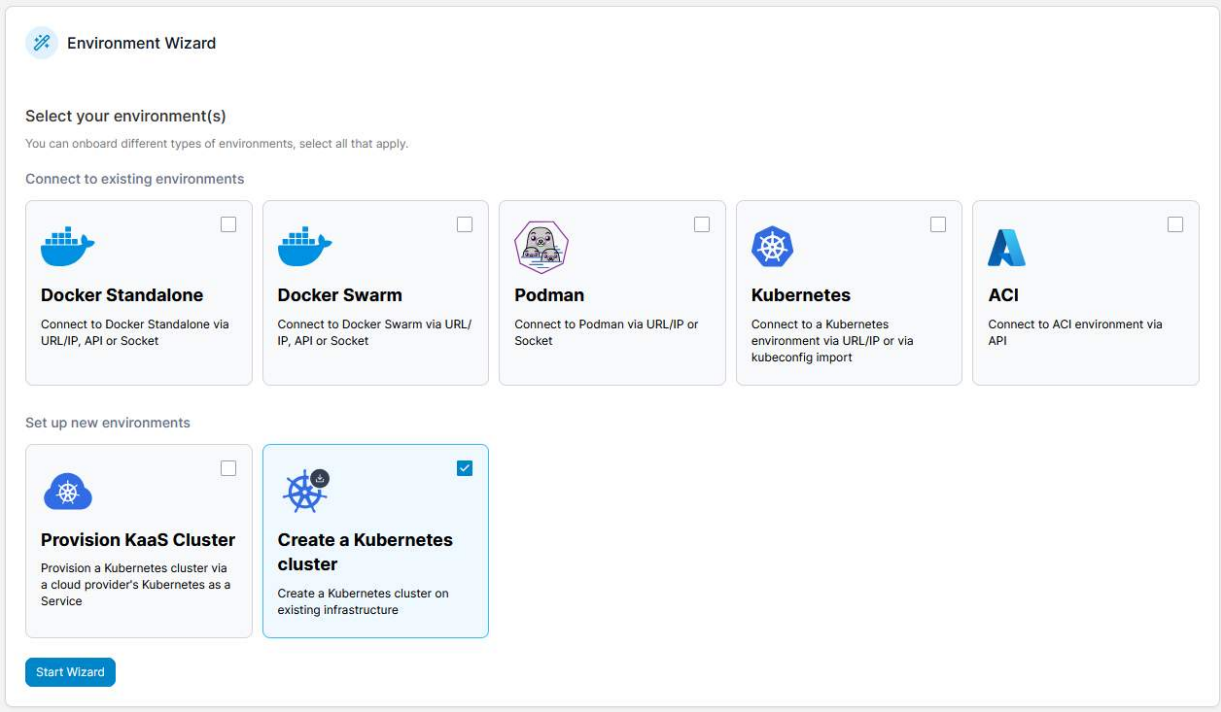
START

Adding a new environment

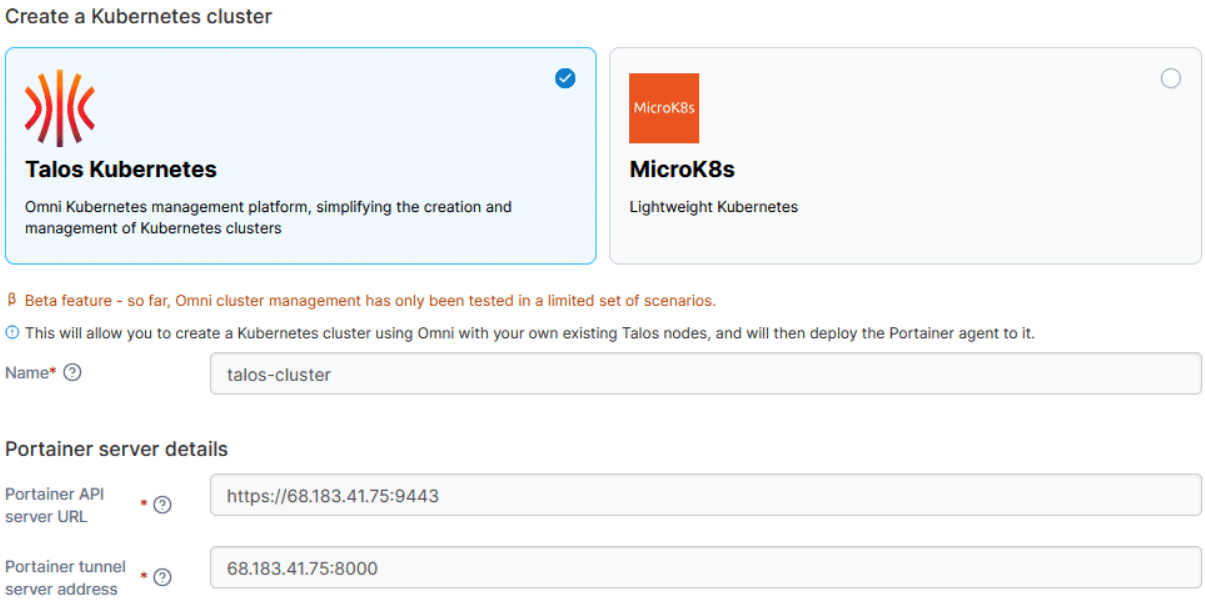
Log into Portainer as the local administrator. Expand the **Environment-related** menu on the left and select **Environments**. Here you'll see a list of the environments configured in Portainer - right now you'll only have the `local` environment where Portainer itself is running.



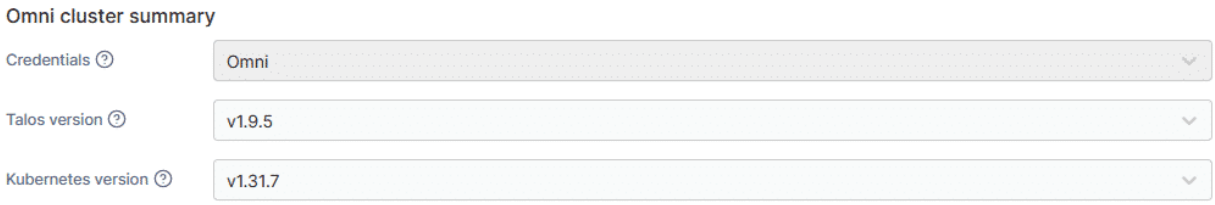
To start provisioning our cluster, click the **Add environment** button. Check the **Create a Kubernetes cluster** option in the Environment Wizard and click **Start Wizard**.



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.

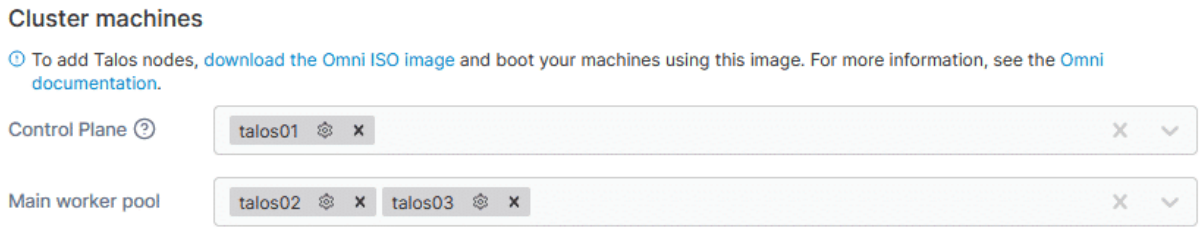


In the **Omni cluster summary** section, your Omni credential set should be automatically selected. For the **Talos version** we can keep it at the default value, but for **Kubernetes version** we'll change this to an older version - for this example, let's choose `v1.31.7`.



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.



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/dow
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

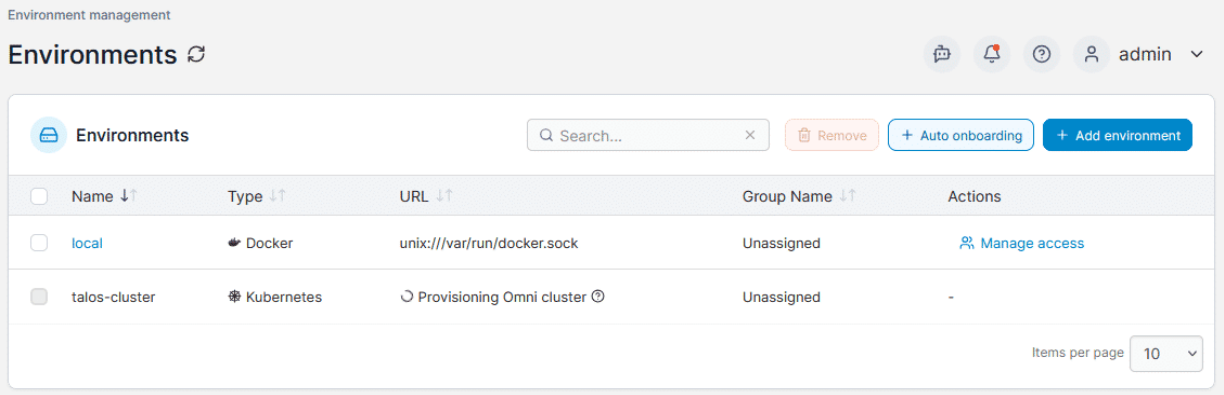
Cluster Configuration Patches allow you to customize settings for all nodes in a cluster or specific groups (such as control plane or worker nodes). See the Talos Config Reference for more information.

Copy to clipboard

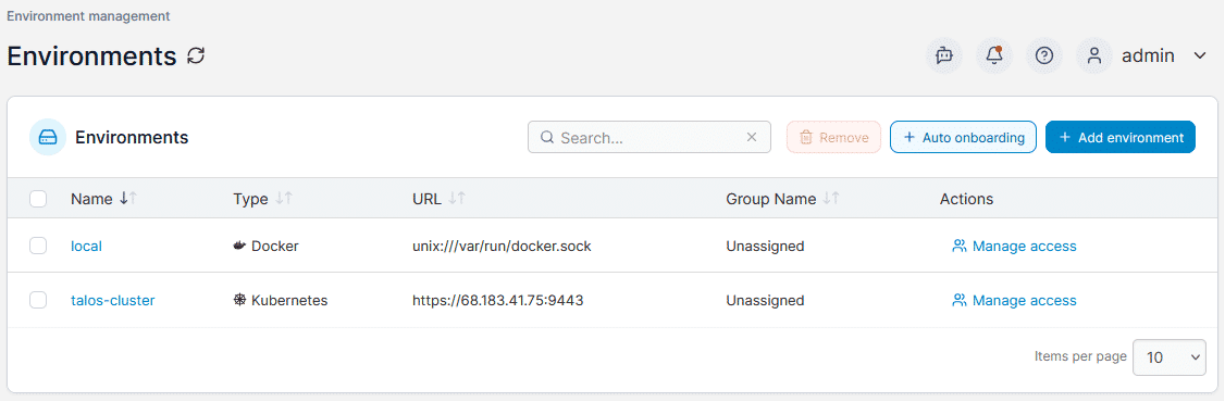
```
cluster:
  network:
    cni:
      flannel:
        extraArgs:
          - --iface-can-reach=10.106.0.4
      name: flannel
  extraManifests:
    - https://raw.githubusercontent.com/alex1989hu/kubelet-serving-cert-approver/main/deploy/standalone-install.yaml
    - https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
machine:
  kubelet:
    extraArgs:
      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.

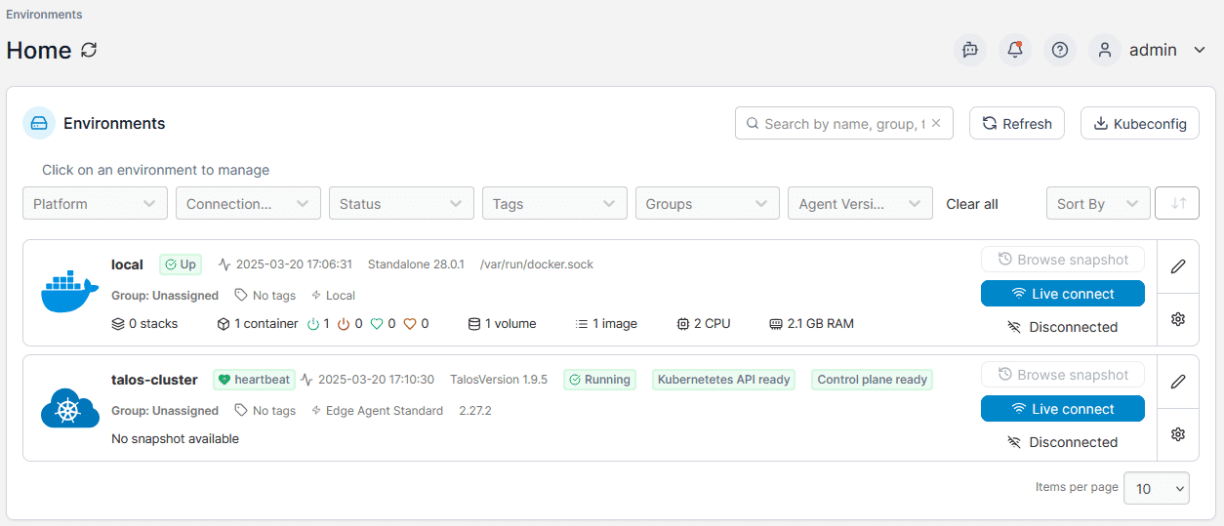
You'll be returned to the **Environments** page, where you should now see your cluster provisioning.



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.



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



CONTINUE

2

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.

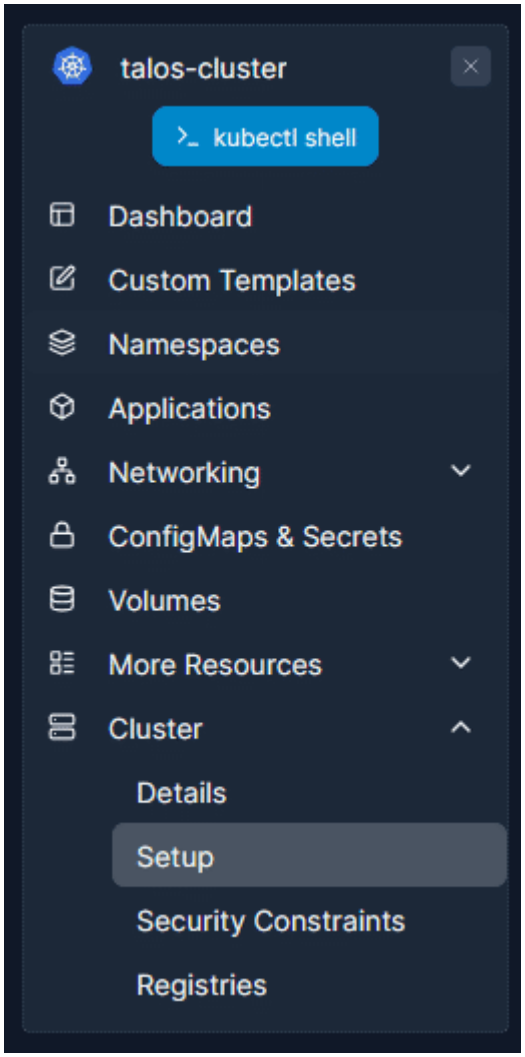
START

Restricting access to the default namespace

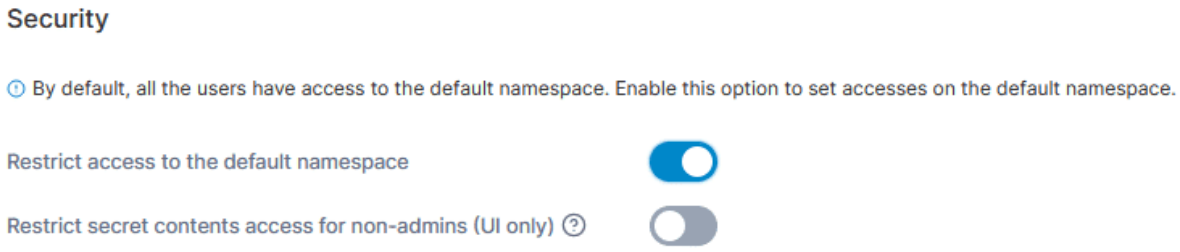
Kubernetes clusters ship with a default namespace configured, which is where workloads that do not specify a namespace would be deployed. For organizational purposes as well as security reasons, we recommend creating namespaces for your workloads. Namespaces can be configured with access restrictions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



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

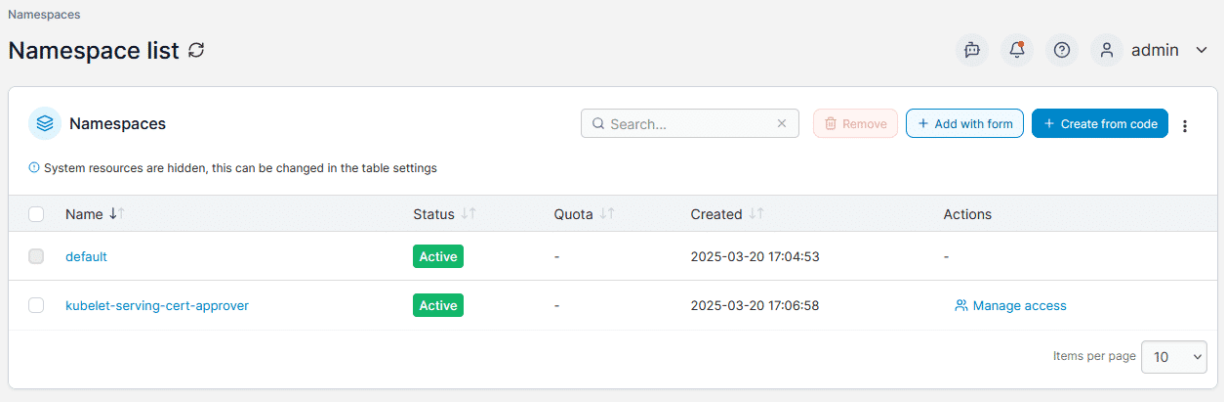
CONTINUE

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.

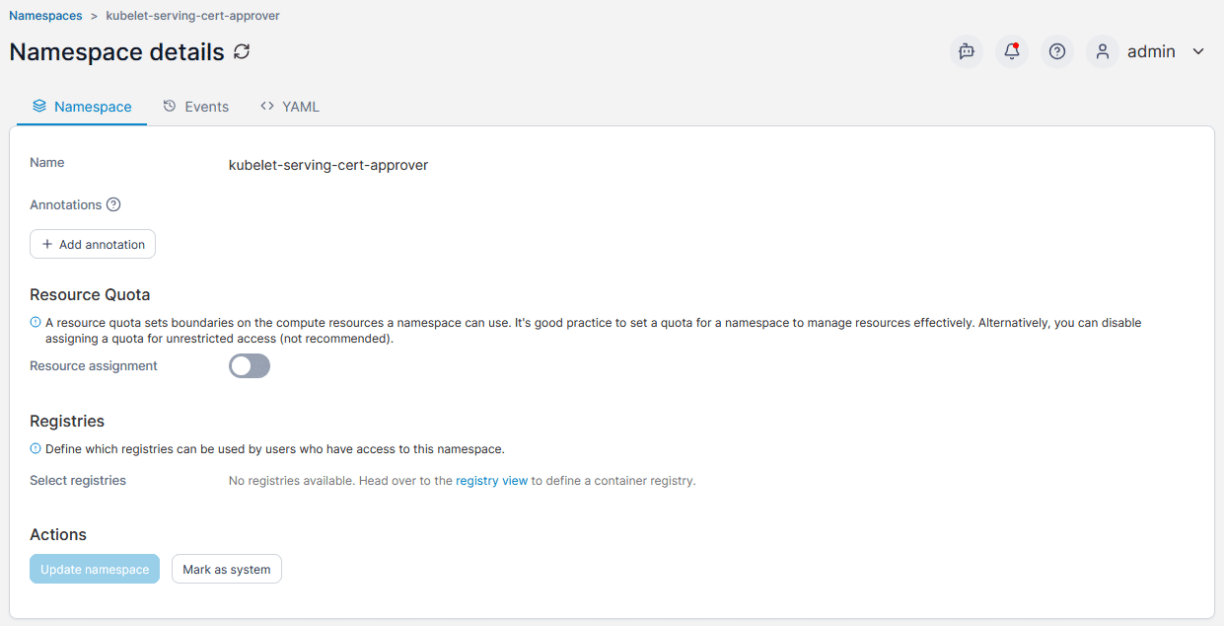
First, we'll need to configure resource quotas on our existing namespaces. With your Kubernetes environment selected, click on **Namespaces** in the left menu. This will take you to the list of namespaces on your environment.



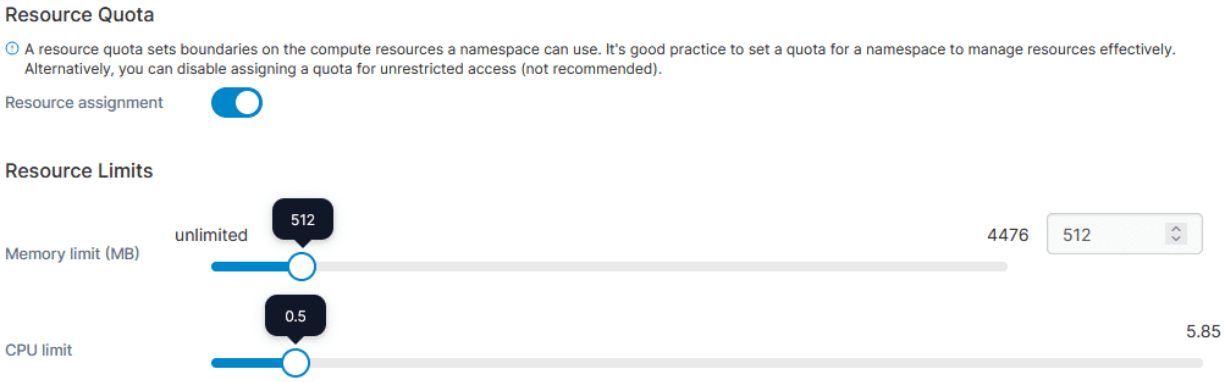
At this point you should have only two namespaces:

- 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.

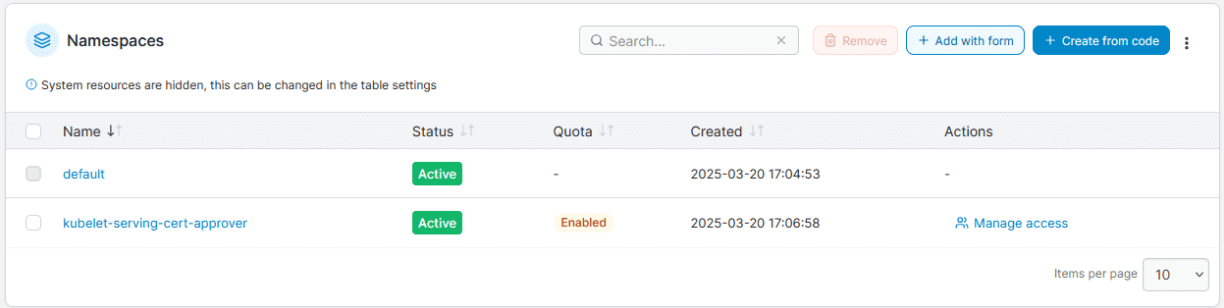
We want to set a quota on this second namespace, so click on `kubelet-serving-cert-approver` to be taken to the **Namespace details** page.



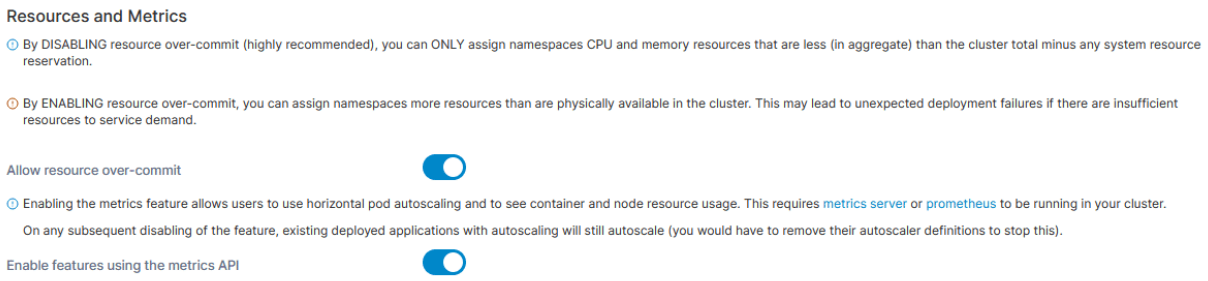
Here you can see the details for this namespace. Under **Resource Quota**, enable the **Resource assignment** toggle. This will display two new options for Memory limit (MB) and CPU limit. For this resource, we're going to set these to values of 512 `MB` for memory and `0.5` for CPU. You can use the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



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.



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%.

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

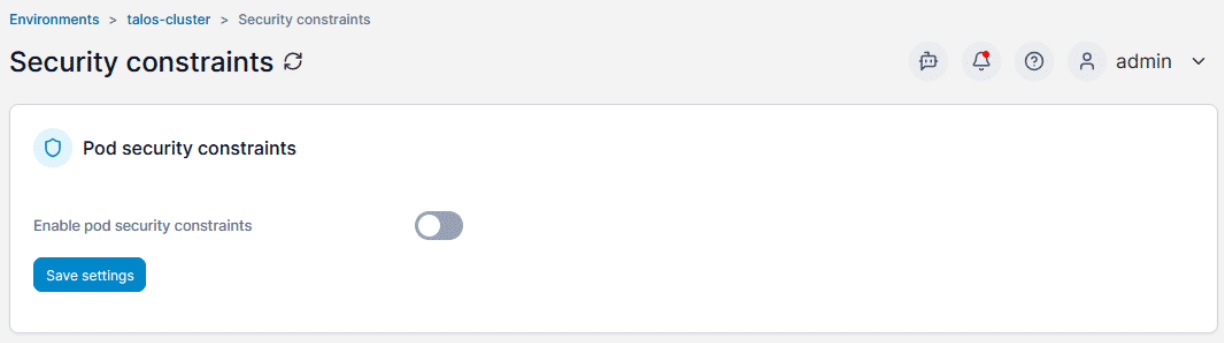
CONTINUE

3

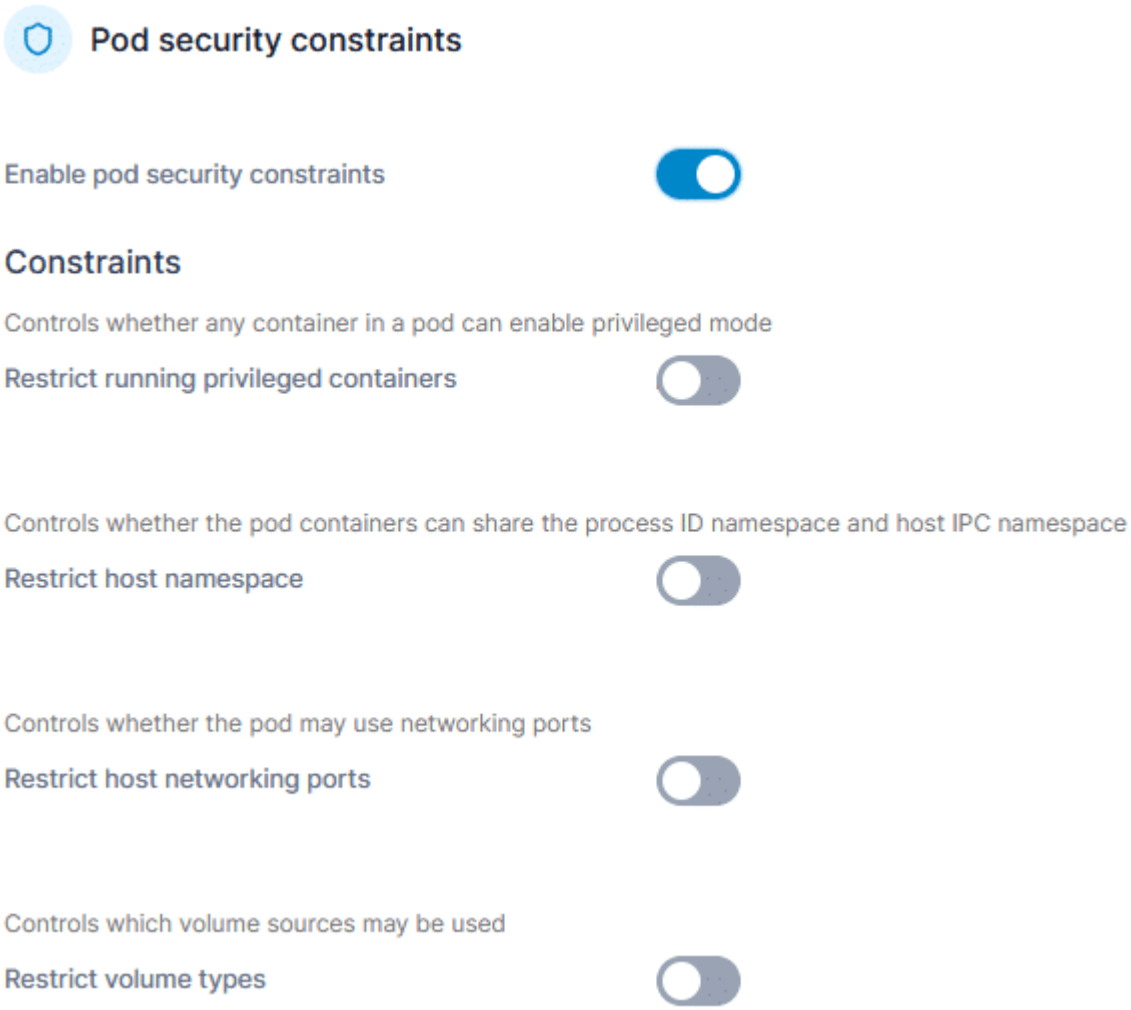
OPA Gatekeeper

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.

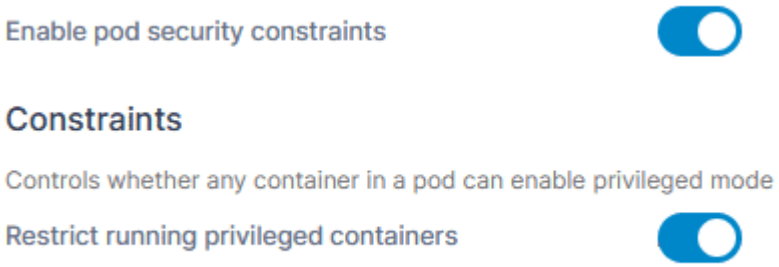
With your Kubernetes environment selected, expand the **Cluster** menu on the left and select **Security Constraints**.



Toggle on the **Enable pod security constraints** option, and a list of possible security constraints will be shown.



A full description of each option is available in our documentation. For now, let's enable **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.

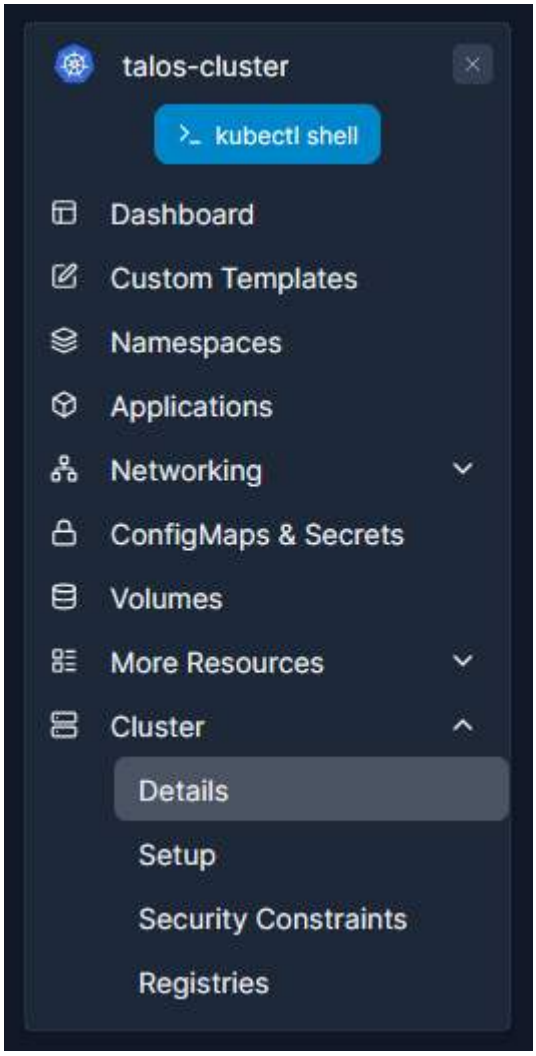
In this lesson we will upgrade our Kubernetes cluster to the latest version.

START

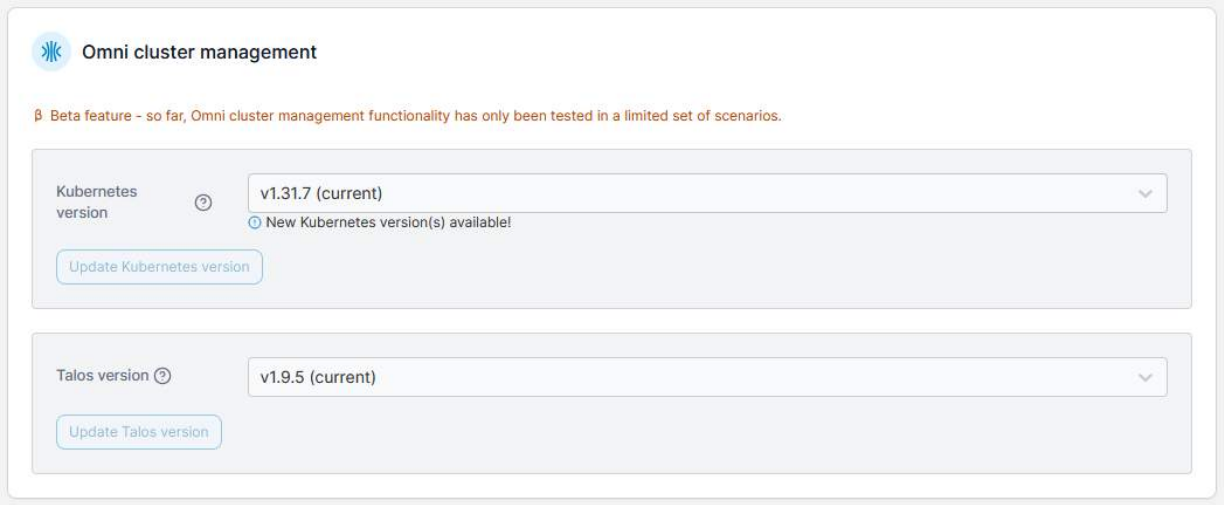
1

Upgrading Kubernetes

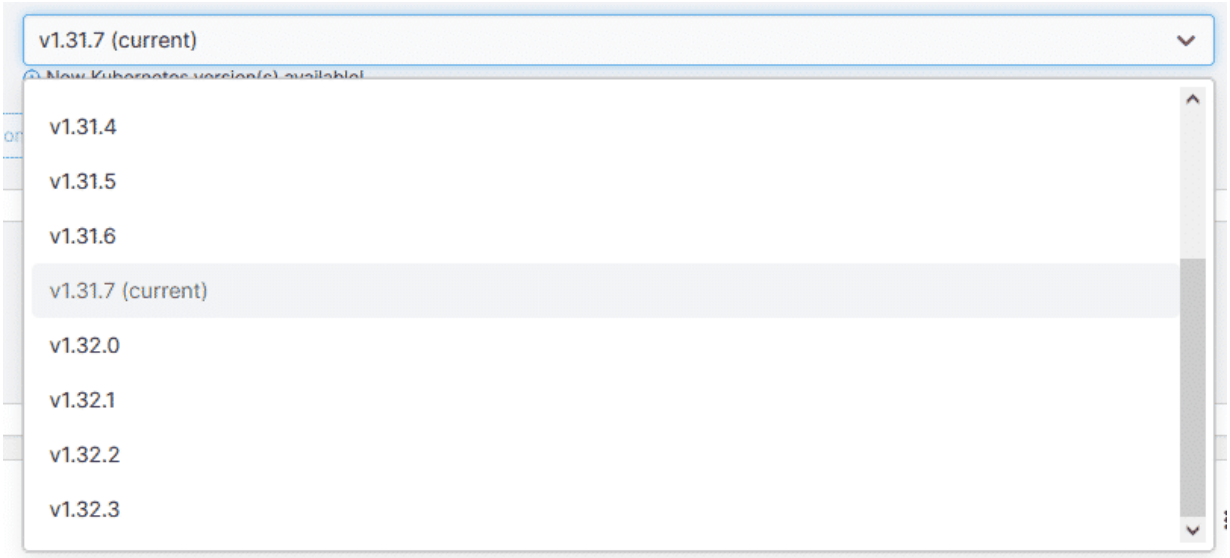
Log into Portainer as the local administrator. Select your Kubernetes environment then expand the **Cluster** menu on the left and select **Details**.



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.

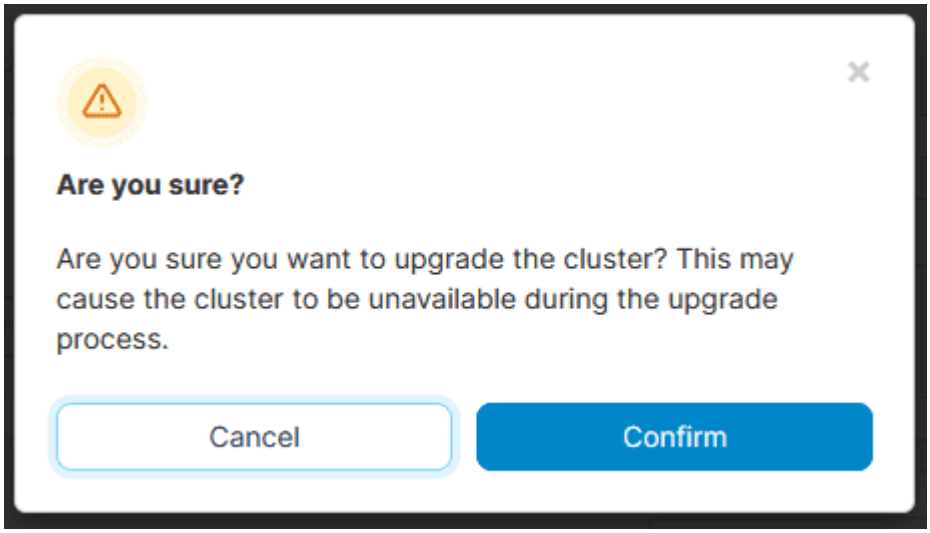


Here we can see the current Kubernetes and Talos versions. When we provisioned the cluster we chose an older version of Kubernetes (`1.31.7`) which is listed as current, and you will note the message indicating a new Kubernetes version is available. If we click the **Kubernetes version** dropdown we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



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

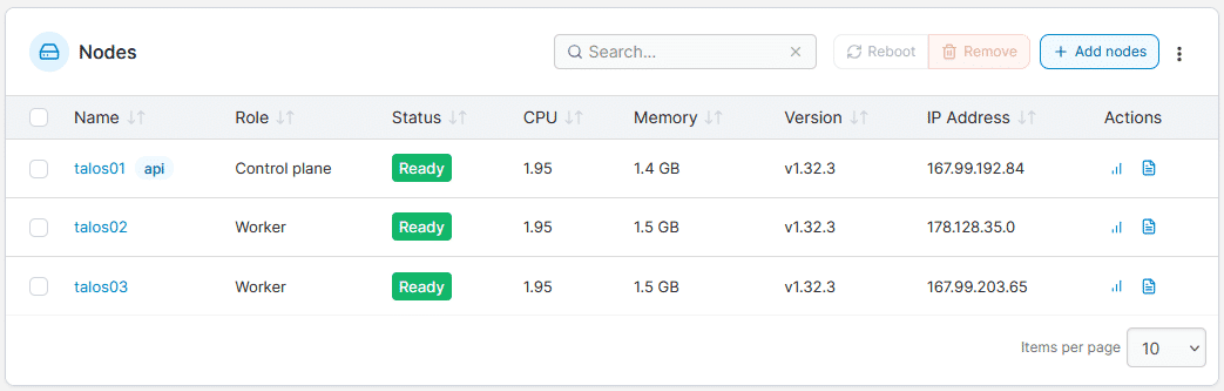
that the cluster may become unavailable during this upgrade - since we're not running any workloads yet, this is fine. Click **Confirm** to proceed.



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.

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.



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:

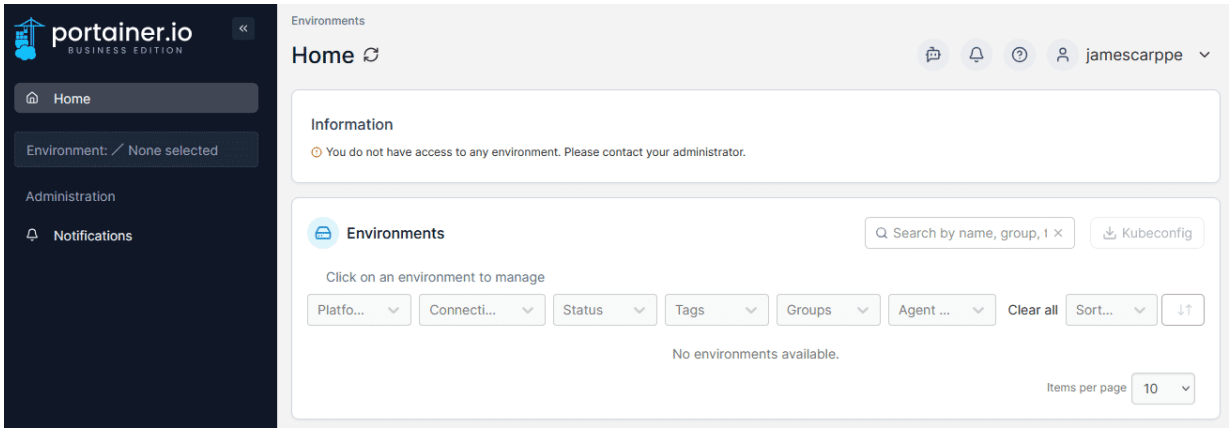
1. Configure our GitHub OAuth user with access to our Kubernetes environment.
2. Create a namespace on the environment and give our GitHub OAuth user access to it.
3. Log in with our GitHub OAuth user and confirm our level of access is as expected.

START

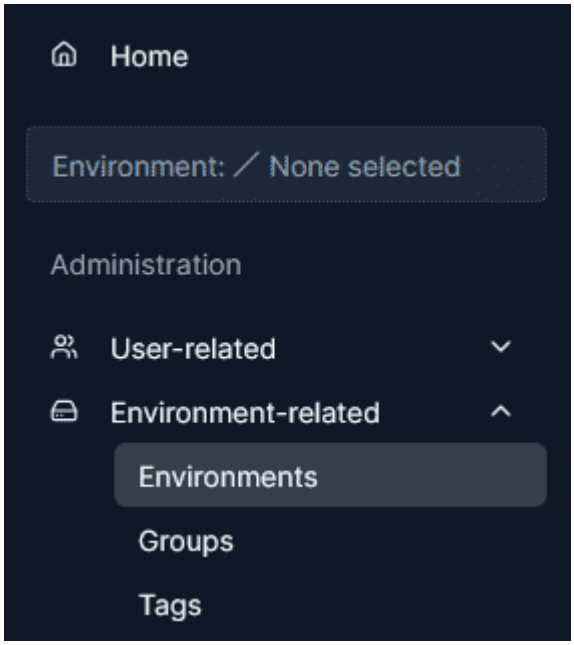
1

Giving our user environment access

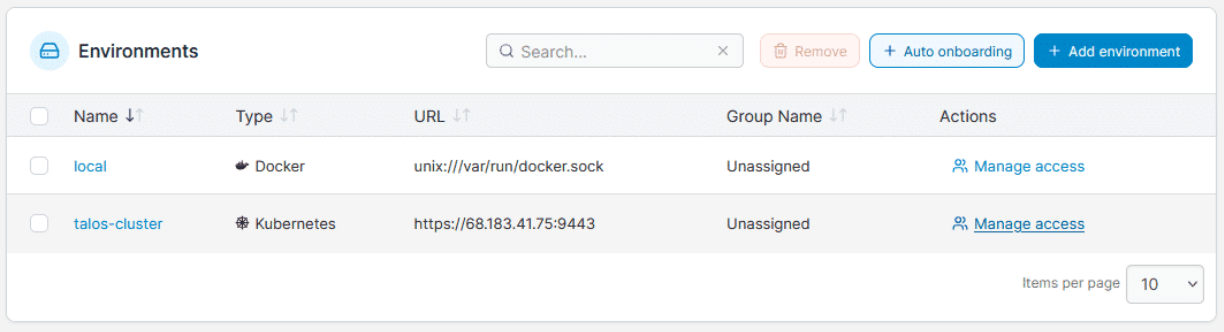
To start with, we need to give our user access to the new Kubernetes environment. As you'll recall from when we configured OAuth, the GitHub user could not see any environments when logged in.



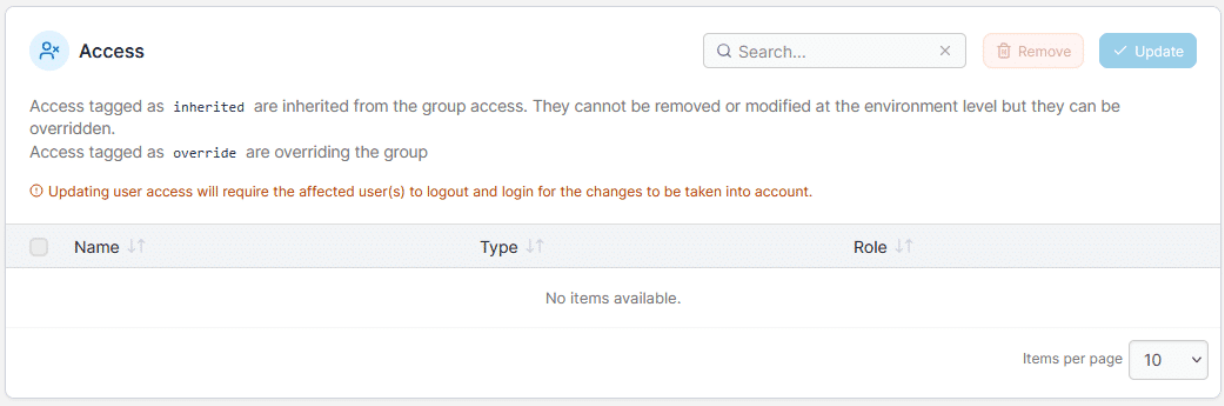
Let's remedy that. Log into Portainer as the local administrator, then in the left menu expand **Environment-related** and select **Environments**.



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`).

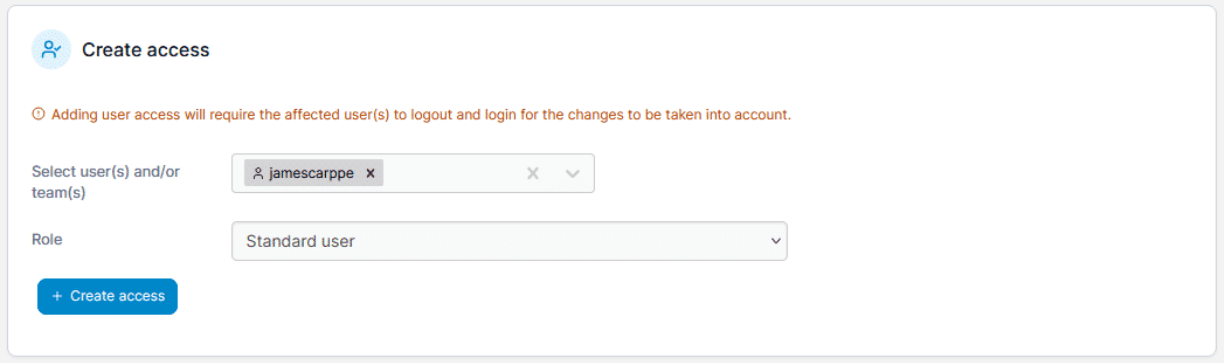


Next to your Kubernetes environment, click the **Manage access** link. Here you can manage access to this environment. Scroll down to the **Access** section and you will see that there are no users or groups listed with access to the environment currently (the initial administrator has access to everything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

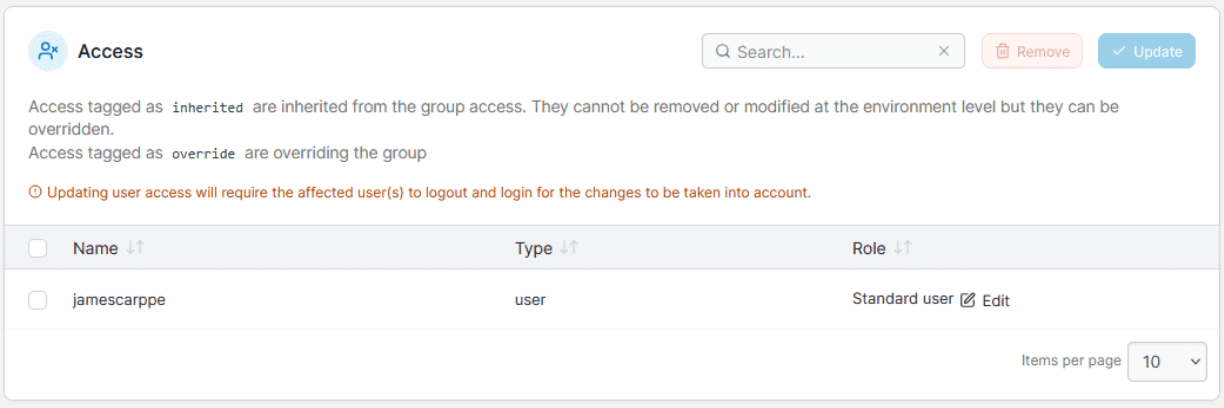


Scroll back up to the **Create access** section and from the **Select user(s) and/or team(s)** dropdown choose your GitHub user. In the **Role** dropdown, choose the `Standard user` role.

> **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.**



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.



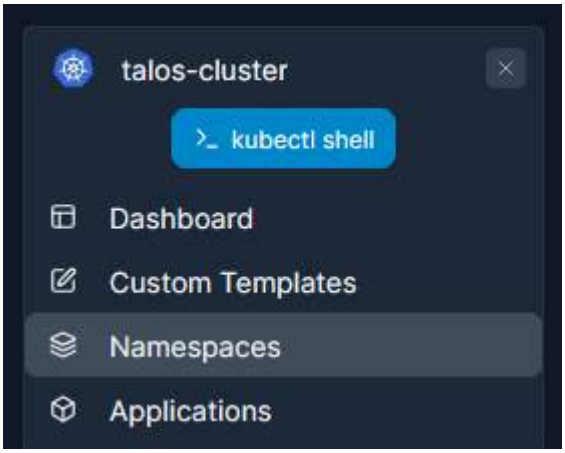
CONTINUE

2

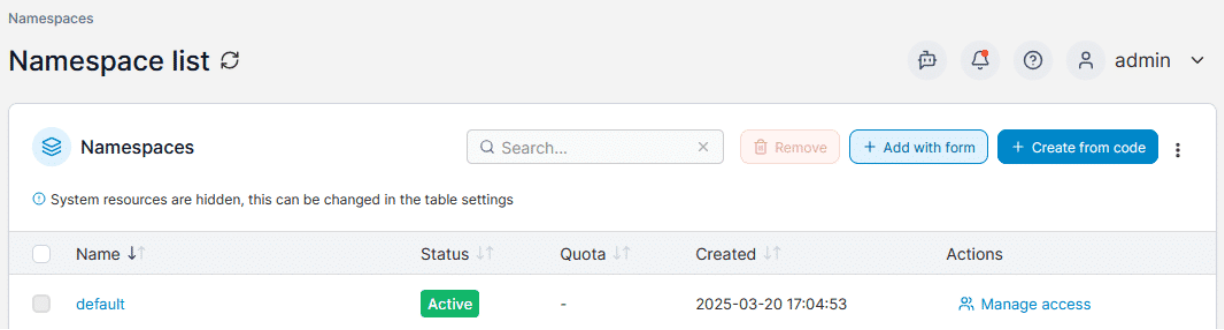
Creating a namespace for the user

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.

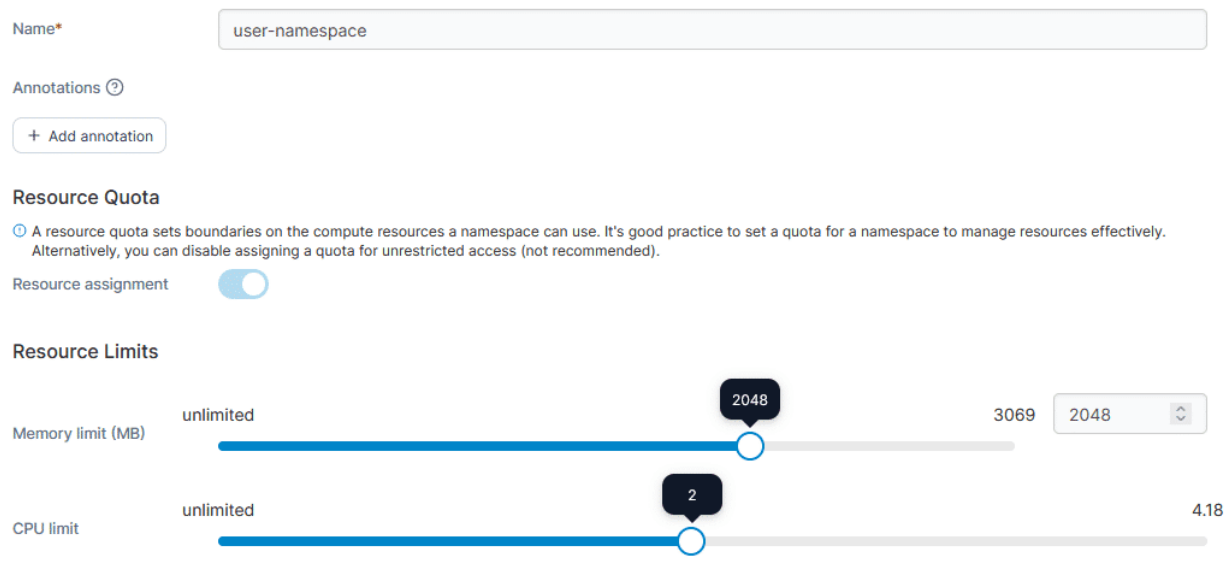
Select your Kubernetes environment then select **Namespaces** from the left menu.



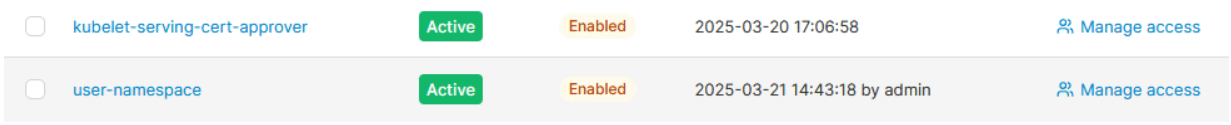
In the **Namespace list** you'll see our existing namespaces. Click on **Add with form** to create a new namespace.



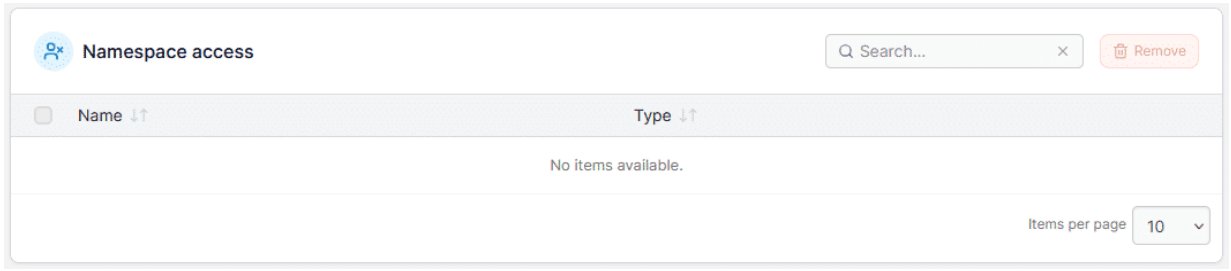
In the **Create a namespace** form, set a **Name** for the namespace. You will also note that we must set memory and CPU limits because we disabled over-commit earlier. Let's give this namespace a memory limit of `2048 MB` and a 2 CPU limit.



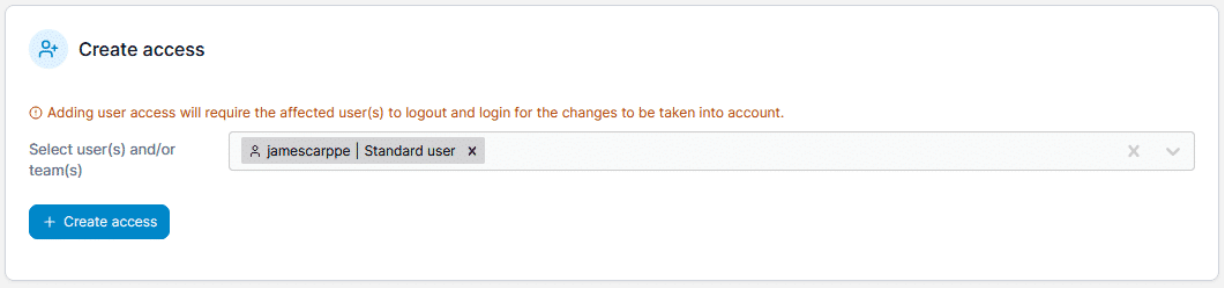
When you're ready, click the **Create namespace** button. The namespace will be created and you'll be returned to the namespace list, where you'll see your new namespace listed.



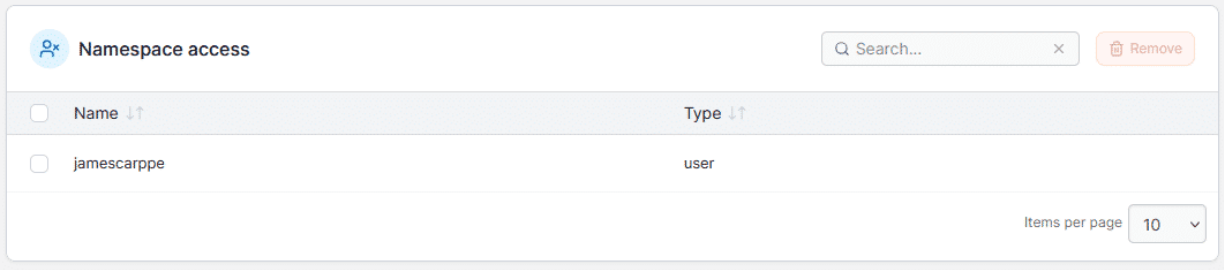
Now we need to give our user access to the namespace. Click on **Manage access** for the namespace we just created. Similar to the environment access management page, this page lists who has access to the namespace (currently nobody but administrators).



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.



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



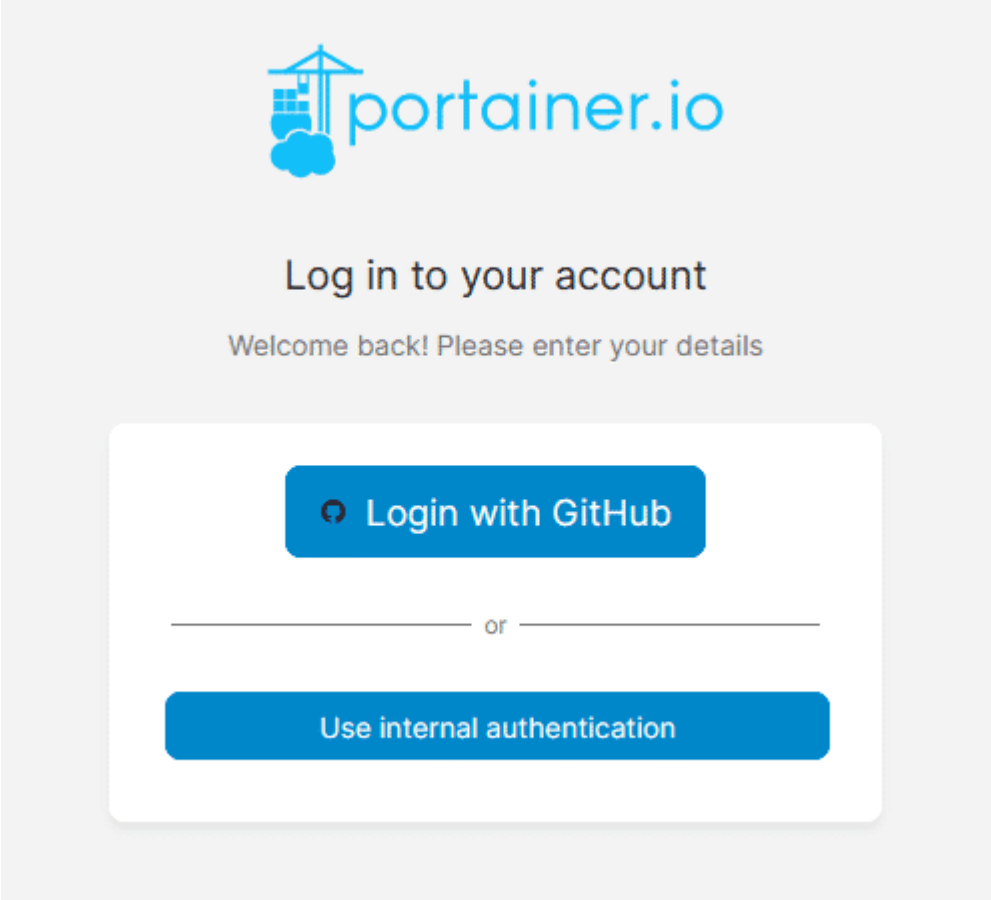
Now let's make sure that all worked, and log in as the GitHub user.

CONTINUE

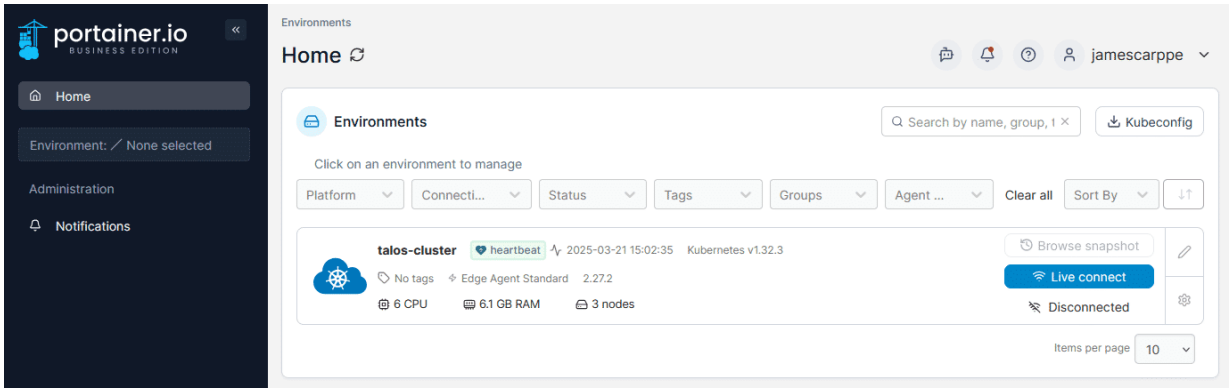
3

Testing access

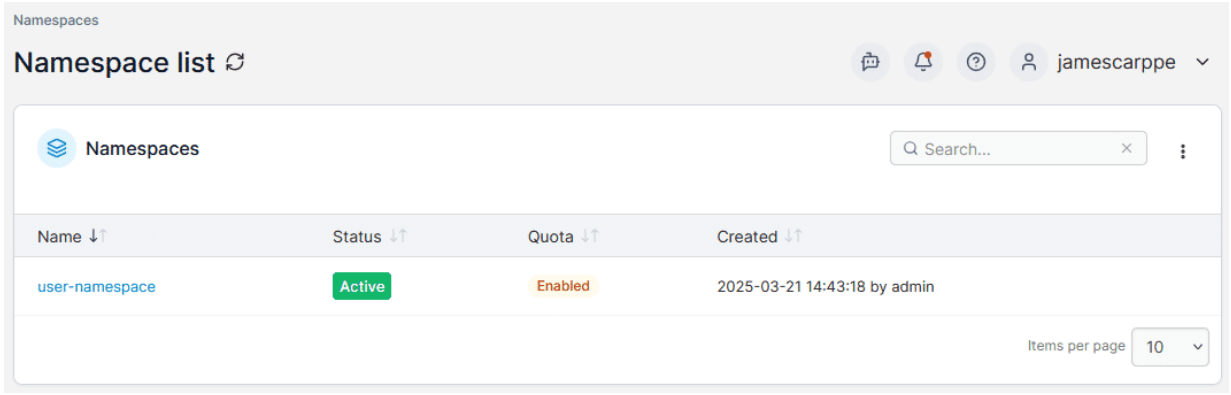
Log out of Portainer (click the arrow next to your username in the top right and select **Log out**) and log back in as the GitHub user by clicking **Login with GitHub** and providing the credentials if necessary.



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

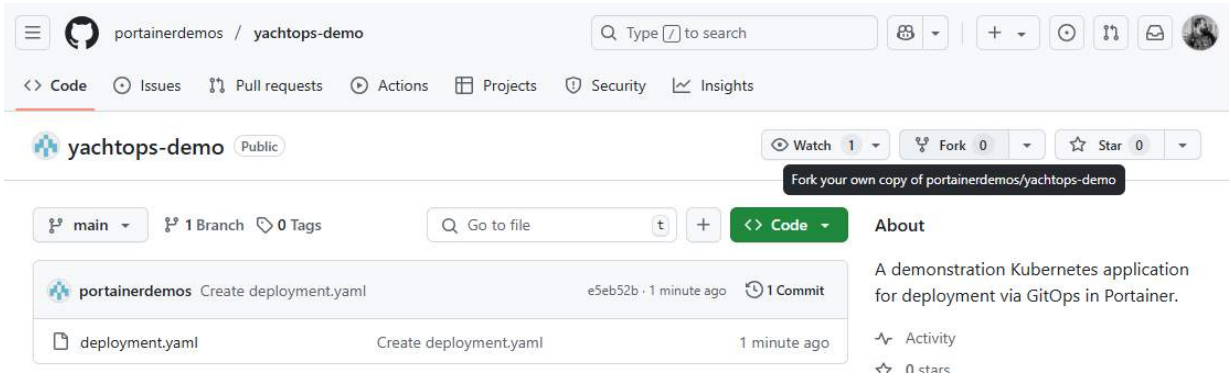
There are a number of methods you can use to deploy an application on a Kubernetes cluster. In this example, we'll demonstrate how you can use a Git repository containing a YAML manifest to deploy an application through Portainer.

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.

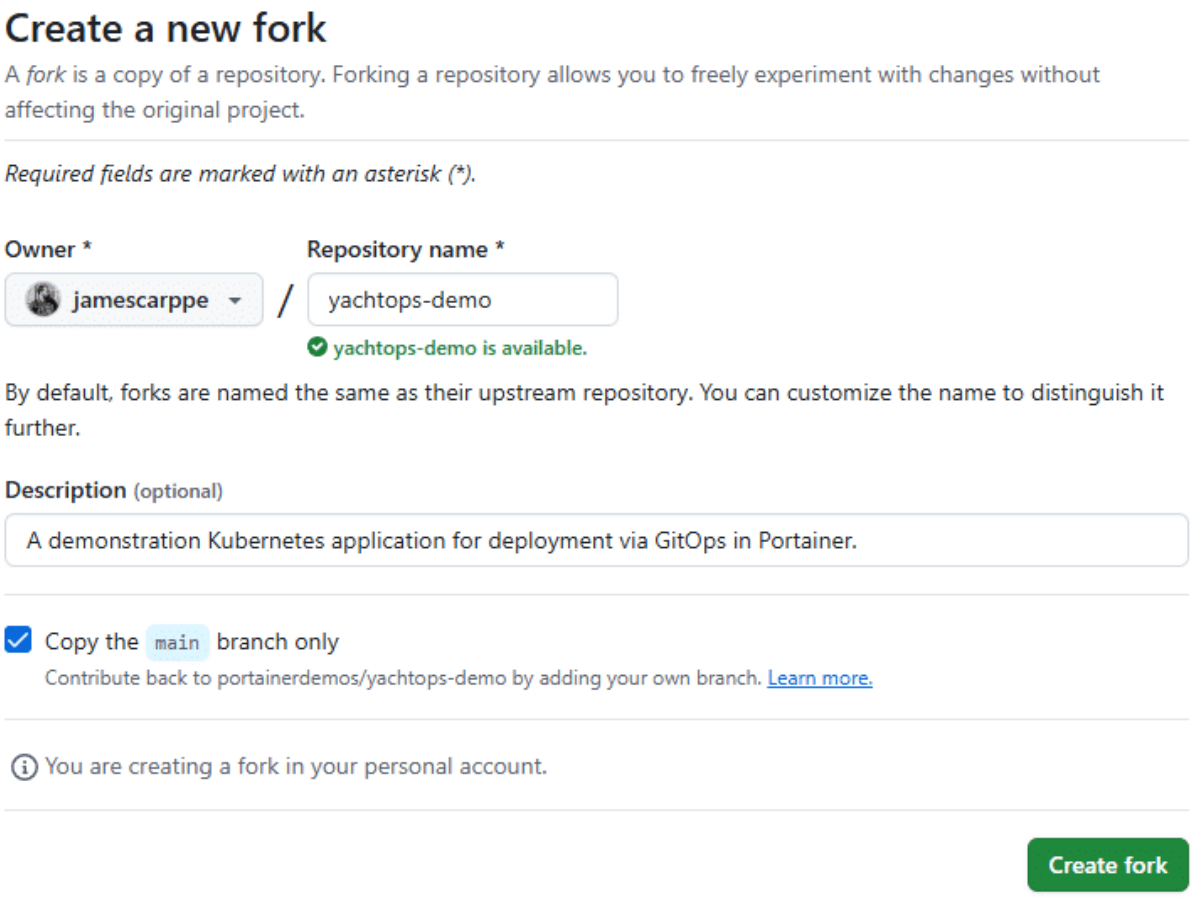
You can find the example repository at the following URL:

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

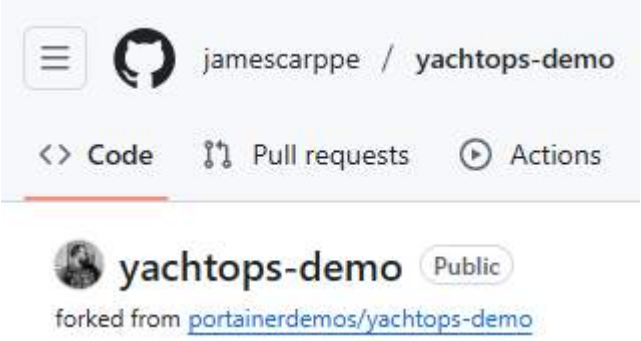
Go to the above repository in a web browser and click the **Fork** button.



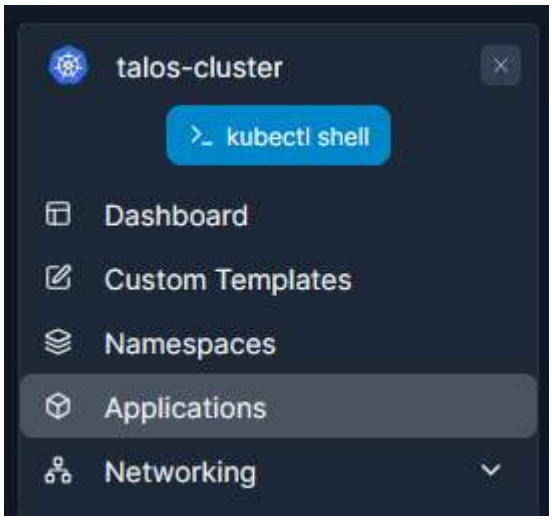
Choose from the list of owners to fork to (if you have access to multiple owners) and edit the name of the repo if you so desire, then click **Create fork**.



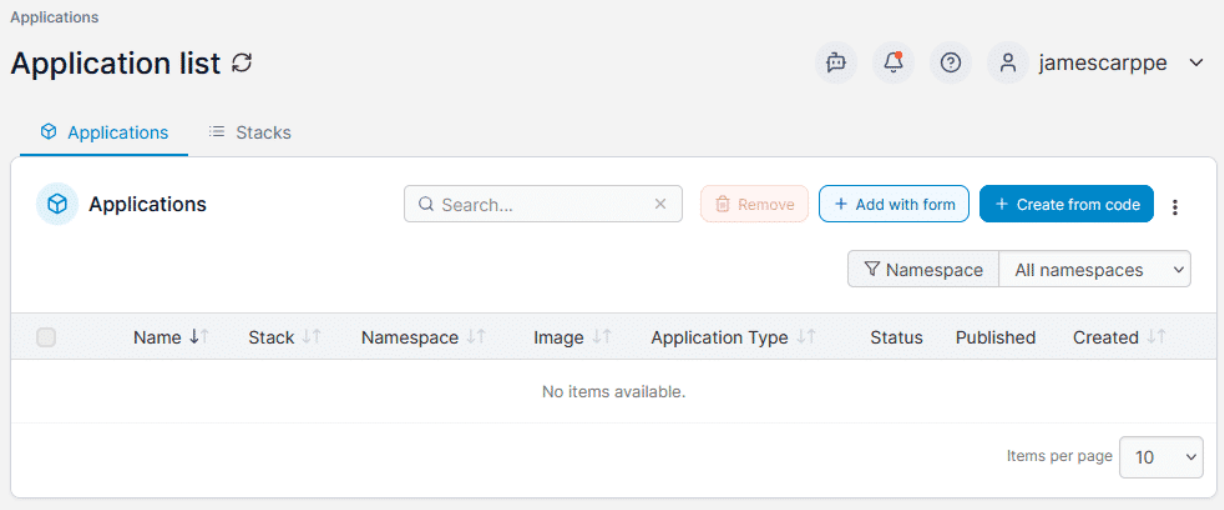
Once the fork has been created you'll have a copy of the repo in your own GitHub account. We'll deploy our application from there.



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

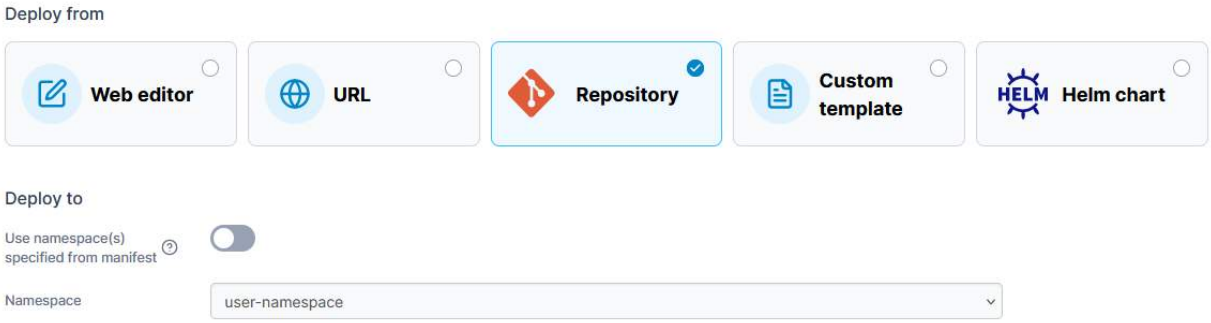


In the **Applications list**, you can see we have no applications currently deployed on the cluster that this user can access. You can use the **Namespace** dropdown to filter the list by namespace, or show all namespaces.

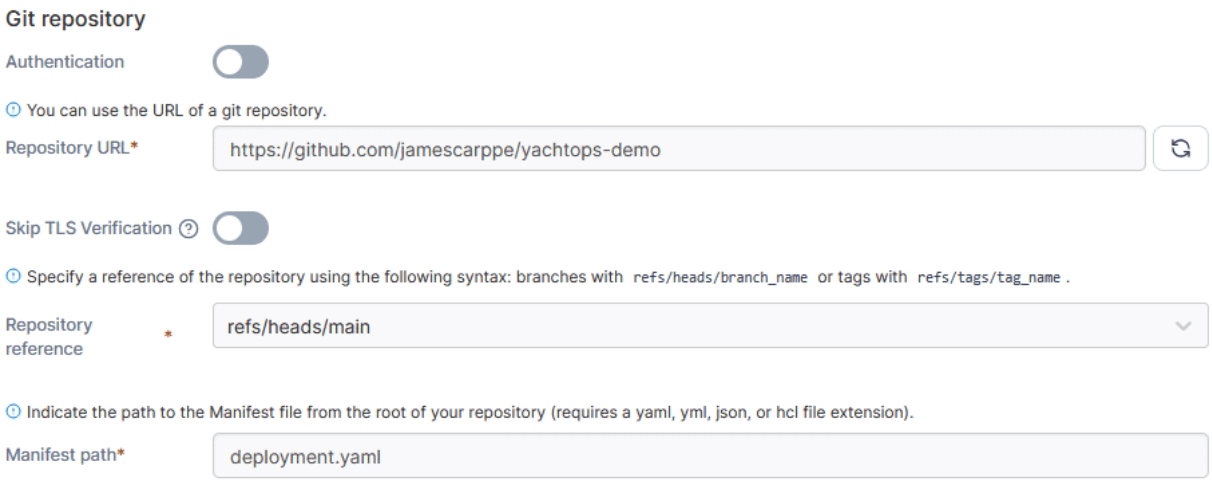


To create our application, click on the **Create from code** button.

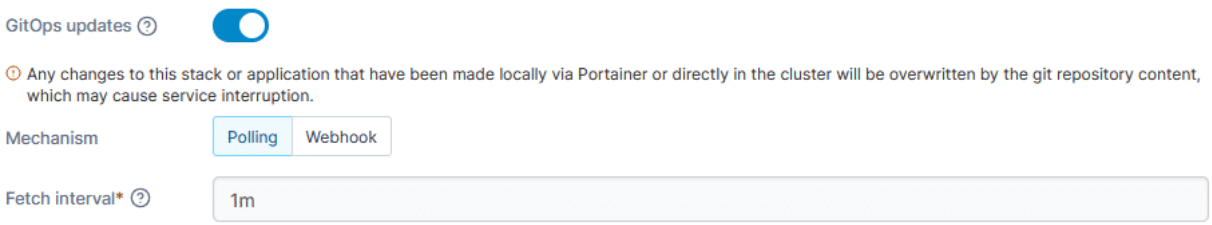
On the **Create from code** page, ensure that **Repository** is selected and that the namespace we created earlier is selected in **Namespaces**.



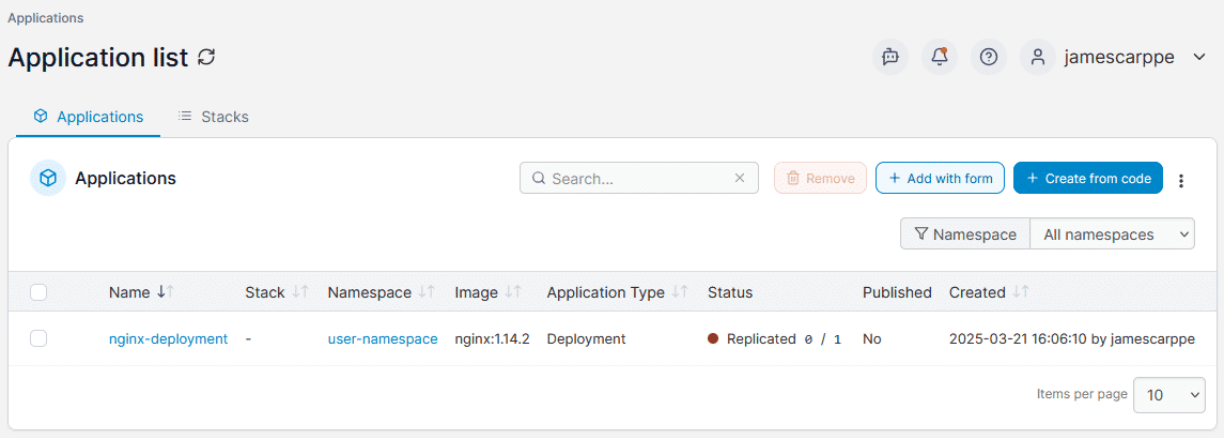
As we're deploying from a Git repository, we need to specify the details. This repository is public, so we don't need to specify authentication. Provide the URL of your forked repository in the **Repository URL** field. The **Repository reference** should automatically populate, and enter `deployment.yaml`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



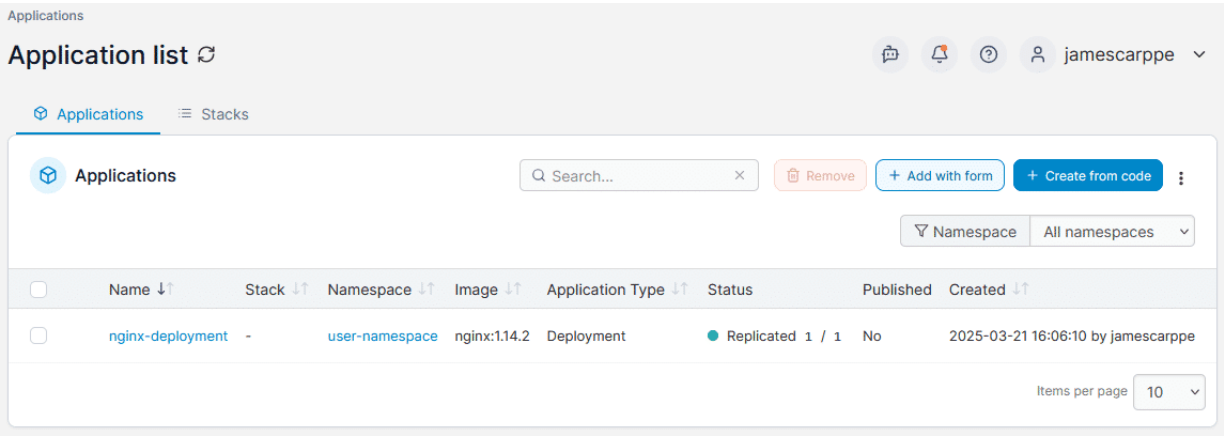
At the bottom of the form, toggle on **GitOps updates**. Here we can configure how Portainer checks for updates to your deployment. Ensure **Polling** is selected as the mechanism and set the **Fetch interval** to `1m`.



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.



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.

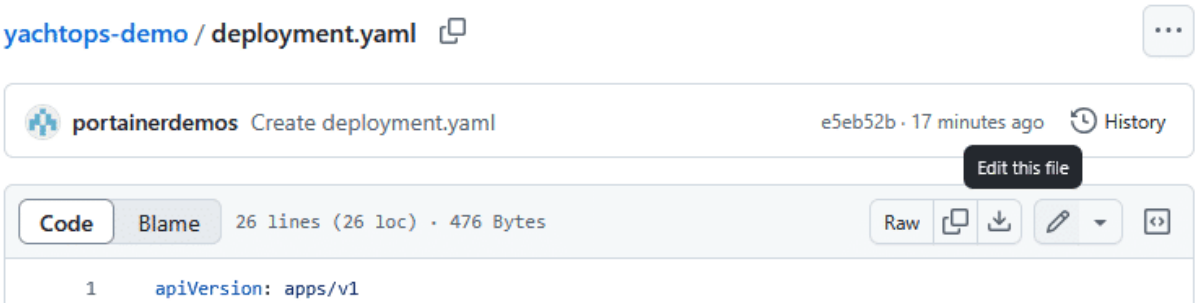


CONTINUE

Making a change to the application

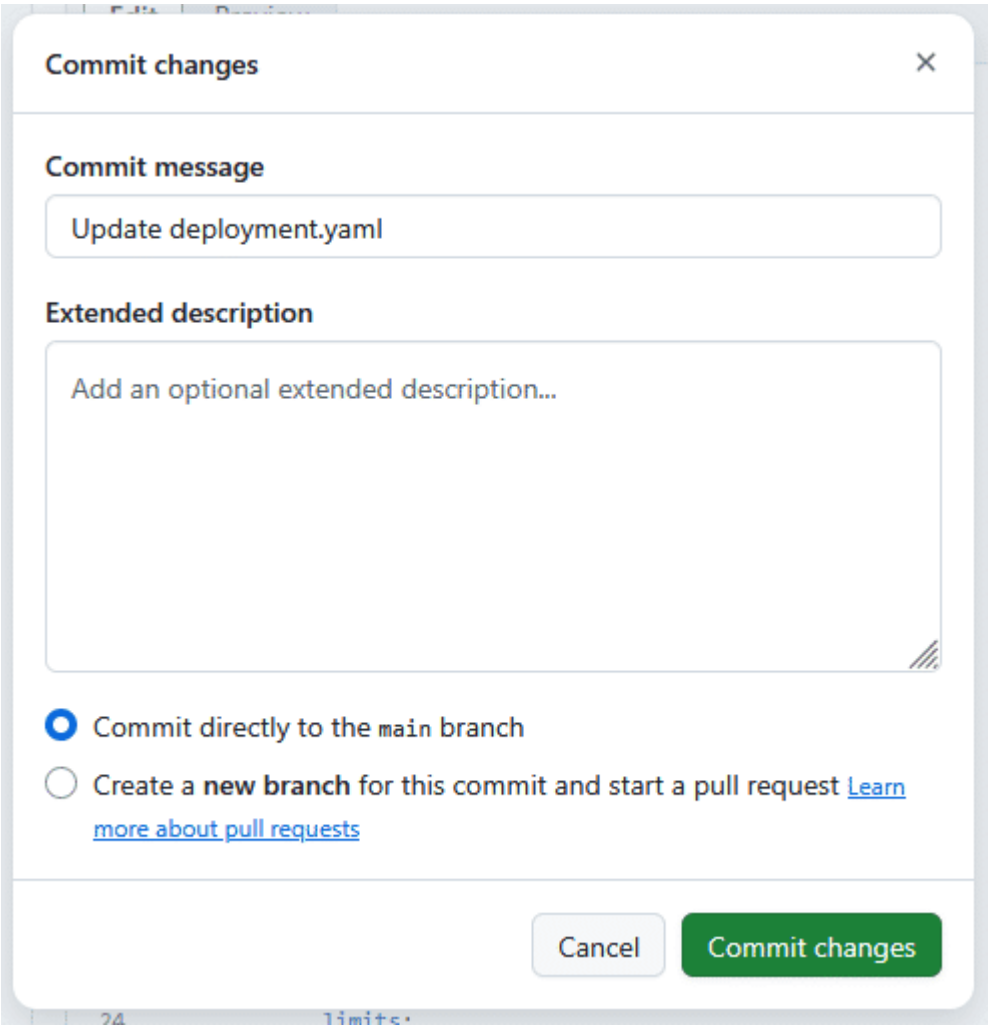
The application has deployed successfully to your cluster. But let's say we wanted to make a change. With the way we've configured this application, all we need to do is update the manifest in the Git repository and Portainer will automatically pick up the change and update the deployment.

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`.

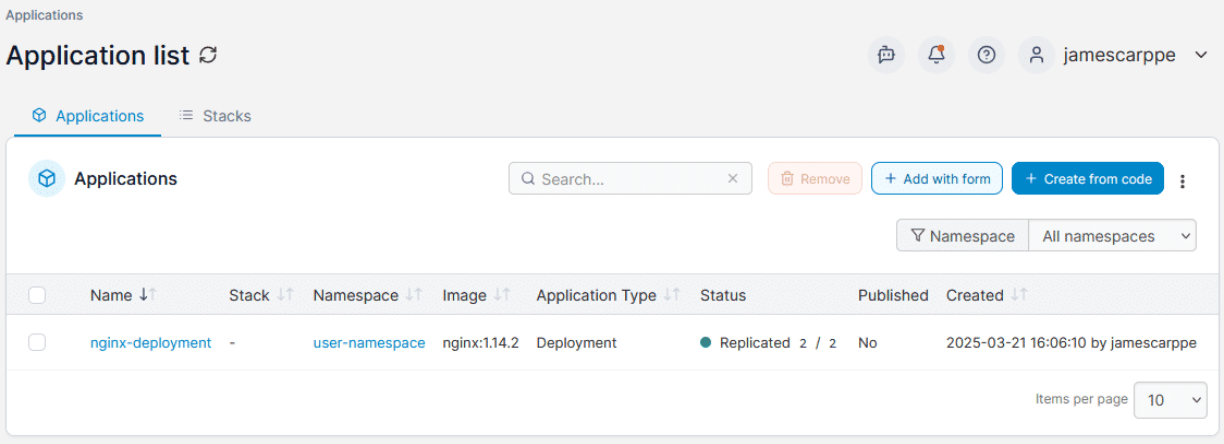
With this changed, click the **Commit changes** button and click **Commit changes** again in the popup window.



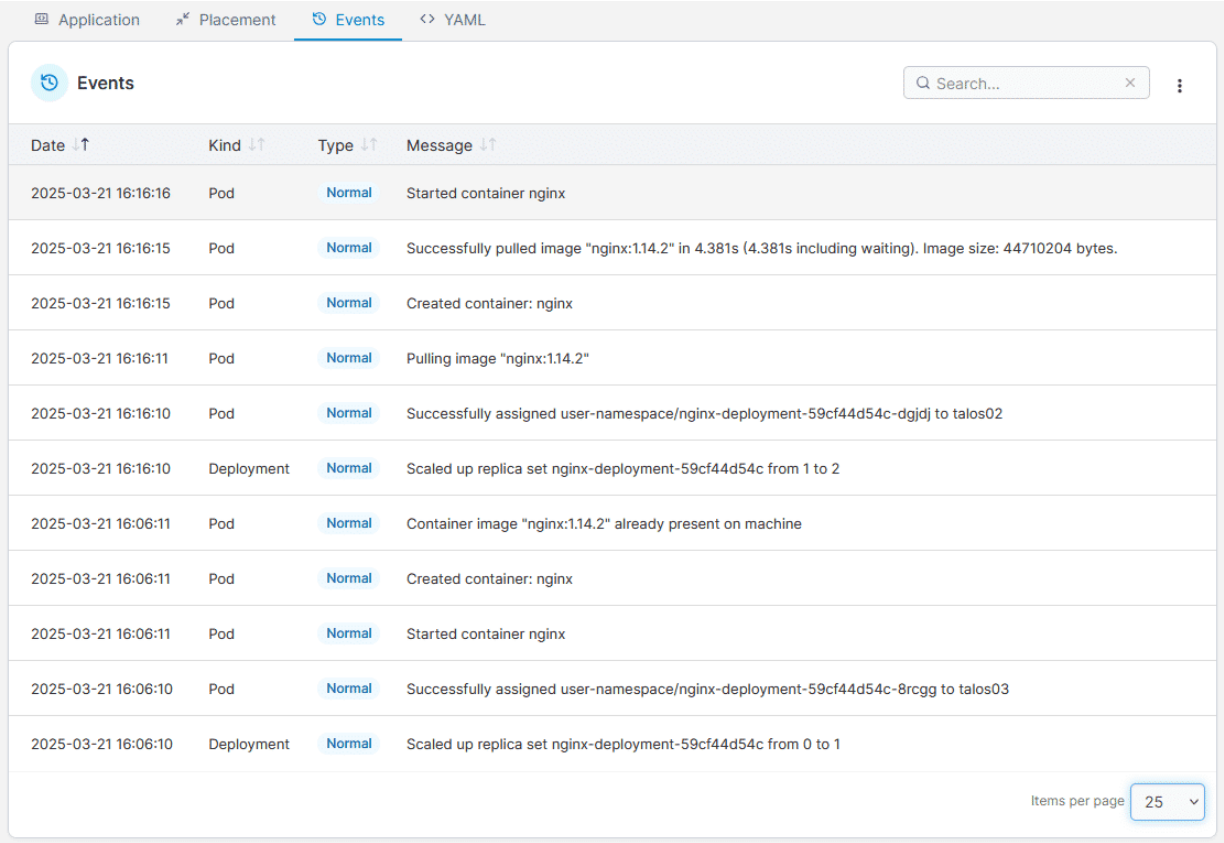
Your changes will be committed to the repository.

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:



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.



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!