

---

# **BinderHub Documentation**

***Release 0.1.0***

**Yuvi Panda**

**Feb 21, 2020**



---

## Contents

---

<b>1</b>	<b>Getting started</b>	<b>3</b>
<b>2</b>	<b>BinderHub Deployments</b>	<b>5</b>
<b>3</b>	<b>Zero to BinderHub</b>	<b>7</b>
3.1	Zero to BinderHub . . . . .	7
<b>4</b>	<b>Customization and deployment information</b>	<b>21</b>
4.1	Customization . . . . .	21
<b>5</b>	<b>BinderHub Developer and Architecture Documentation</b>	<b>31</b>
5.1	Administrators and developers . . . . .	31
<b>6</b>	<b>The BinderHub community</b>	<b>49</b>
6.1	Community . . . . .	49
	<b>Python Module Index</b>	<b>53</b>
	<b>Index</b>	<b>55</b>







# CHAPTER 1

---

## Getting started

---

The primary goal of BinderHub is creating custom computing environments that can be used by many remote users. BinderHub enables an end user to easily specify a desired computing environment from a Git repo. BinderHub then serves the custom computing environment at a URL which users can access remotely.

This guide assists you, an administrator, through the process of setting up your BinderHub deployment.

To get started creating your own BinderHub, start with [Zero to BinderHub](#).

---

**Note:** BinderHub uses a JupyterHub running on Kubernetes for much of its functionality. For information on setting up and customizing your JupyterHub, we recommend reading the [Zero to JupyterHub Guide](#).

---





## CHAPTER 2

---

### BinderHub Deployments

---

Our directory of BinderHubs is published at *BinderHub Deployments*.

If your BinderHub deployment is not listed, please [open an issue](#) to discuss adding it.



---

## Zero to BinderHub

---

A guide to help you create your own BinderHub from scratch.

### 3.1 Zero to BinderHub

A guide to help you create your own BinderHub from scratch. Each section below covers various aspects of creating cloud resources and setting up a BinderHub for your users.

#### 3.1.1 Create your cloud resources

BinderHub is built to run on top of Kubernetes, a distributed cluster manager. It uses a JupyterHub to launch/manage user servers, as well as a docker registry to cache images.

To create your own BinderHub, you'll first need to set up a properly configured Kubernetes Cluster on the cloud, and then configure the various components correctly. The following instructions will assist you in doing so.

---

**Note:** BinderHub uses a JupyterHub running on Kubernetes for much of its functionality. For information on setting up and customizing your JupyterHub, we recommend reading the [Zero to JupyterHub Guide](#).

---

#### Setting up Kubernetes on Google Cloud

---

**Note:** BinderHub is built to be cloud agnostic, and can run on various cloud providers (as well as bare metal). However, here we only provide instructions for Google Cloud as it has been the most extensively-tested. If you would like to help with adding instructions for other cloud providers, [please contact us](#)!

---

First, install Kubernetes by following the [instructions in the Zero to JupyterHub guide](#). When you're done, move on to the next section.

## Install Helm

**Helm**, the package manager for Kubernetes, is a useful tool for: installing, upgrading and managing applications on a Kubernetes cluster. Helm packages are called *charts*. We will be installing and managing JupyterHub on our Kubernetes cluster using a Helm chart.

Charts are abstractions describing how to install packages onto a Kubernetes cluster. When a chart is deployed, it works as a templating engine to populate multiple `yaml` files for package dependencies with the required variables, and then runs `kubectl apply` to apply the configuration to the resource and install the package.

Helm has two parts: a client (`helm`) and a server (`tiller`). Tiller runs inside of your Kubernetes cluster as a pod in the `kube-system` namespace. Tiller manages both, the *releases* (installations) and *revisions* (versions) of charts deployed on the cluster. When you run `helm` commands, your local Helm client sends instructions to `tiller` in the cluster that in turn make the requested changes.

---

**Note:** These instructions are for Helm 2. Helm 3 includes several major breaking changes and is not yet officially supported, but [preliminary instructions are available for testing](#).

---

## Installation

While several [methods to install Helm](#) exists, the simplest way to install Helm is to run Helm's installer script in a terminal:

```
curl https://raw.githubusercontent.com/kubernetes/helm/master/scripts/get | bash
```

## Initialization

After installing `helm` on your machine, initialize Helm on your Kubernetes cluster:

1. Set up a `ServiceAccount` for use by `tiller`.

```
kubectl --namespace kube-system create serviceaccount tiller
```

2. Give the `ServiceAccount` full permissions to manage the cluster.

---

**Note:** If you know your kubernetes cluster does not have RBAC enabled, you **must** skip this step. Most users can ignore this note.

---

```
kubectl create clusterrolebinding tiller --clusterrole cluster-admin --  
↪serviceaccount=kube-system:tiller
```

See [our RBAC documentation](#) for more information.

3. Initialize `helm` and `tiller`.

```
helm init --service-account tiller --wait
```

This command only needs to run once per Kubernetes cluster, it will create a `tiller` deployment in the `kube-system` namespace and setup your local `helm` client. This command installs and configures the `tiller` part of Helm (the whole project, not the CLI) on the remote kubernetes cluster. Later when you want to deploy changes with `helm` (the local CLI), it will talk to `tiller` and tell it what to do. `tiller` then executes these instructions from within the cluster.

**Note:** If you wish to install `helm` on another computer, you won't need to setup `tiller` again but you still need to initialize `helm`:

```
helm init --client-only
```

## Secure Helm

Ensure that `tiller` is secure from access inside the cluster:

```
kubectl patch deployment tiller-deploy --namespace=kube-system --type=json --patch='[{"op": "add", "path": "/spec/template/spec/containers/0/command", "value": ["/tiller", "--listen=localhost:44134"]}']'
```

`tiller`'s port is exposed in the cluster without authentication and if you probe this port directly (i.e. by bypassing `helm`) then `tiller`'s permissions can be exploited. This step forces `tiller` to listen to commands from `localhost` (i.e. `helm`) *only* so that e.g. other pods inside the cluster cannot ask `tiller` to install a new chart granting them arbitrary, elevated RBAC privileges and exploit them. [More details here.](#)

## Verify

You can verify that you have the correct version and that it installed properly by running:

```
helm version
```

It should in less than a minute, when `tiller` on the cluster is ready, be able to provide output like below. Make sure you have at least version 2.11.0 and that the client (`helm`) and server version (`tiller`) is matching!

```
Client: &version.Version{SemVer:"v2.11.0", GitCommit:"2e55db1fdb5fdb96b75ff144a339489417b146b", GitTreeState:"clean"}
Server: &version.Version{SemVer:"v2.11.0", GitCommit:"2e55db1fdb5fdb96b75ff144a339489417b146b", GitTreeState:"clean"}
```

Now that you've installed Kubernetes and Helm, it's time to *Set up the container registry*.

### 3.1.2 Set up the container registry

BinderHub will build Docker images out of Git repositories, and then push them to a Docker registry so that JupyterHub can launch user servers based on these images. You can use any registry that you like, though this guide covers how to properly configure several popular registries. The next step, *Set up BinderHub*, explains how you can properly configure BinderHub to use one of these registries.

#### Set up Google Container Registry

To use Google Container Registry (`gcr.io`), you'll need to provide BinderHub with proper credentials so it can push images. You can do so by creating a service account that has authorization to push to Google Container Registry:

1. Go to [console.cloud.google.com](https://console.cloud.google.com)
2. Make sure your project is selected

3. Click `<top-left menu w/ three horizontal bars>` -> IAM & Admin -> Service Accounts menu option
4. Click **Create service account**
5. Give your account a descriptive name such as “binderhub-builder”
6. Click Role -> Storage -> Storage Admin menu option
7. Click **Create Key**
8. Leave key type as default of **JSON**
9. Click **Create**

These steps will download a **JSON file** to your computer. The JSON file contains the password that can be used to push Docker images to the `gcr.io` registry.

**Warning:** Don't share the contents of this JSON file with anyone. It can be used to gain access to your google cloud account!

---

**Important:** Make sure to store this JSON file as you cannot generate a second one without re-doing the steps above.

---

### Set up Docker Hub registry

To use **Docker Hub** (`hub.docker.com`) as a registry first you have to create a [Docker ID account in Docker Hub](#). Your Docker ID (username) and password are used to push Docker images to the registry.

If you want to store Docker images under an organization, you can [create an organization](#). This is useful if different Binder instances want to use same registry to store images.

### Set up Azure Container Registry

To use Azure Container Registry (ACR), you'll need to provide BinderHub with proper credentials so it can push images. You can do so by creating a [Service Principal](#) that has the [AcrPush](#) role.

This section uses the Azure command line. Installation instructions can be found in the [Microsoft docs](#).

1. Login to your Azure account:

```
az login
```

2. Select your chosen subscription:

```
az account set -s <SUBSCRIPTION>
```

---

**Note:** You can see which subscriptions you have available by running:

```
az account list --refresh --output table
```

3. If you **do not** have a Resource Group, then create one:

```
az group create --name <RESOURCE_GROUP_NAME> --location <RESOURCE_GROUP_LOCATION>
↪--output table
```

where <RESOURCE\_GROUP\_LOCATION> refers to a data centre **region**. See a list of regions [here](#).

If you already have a Resource Group you'd like to use, then you can skip this step.

#### 4. Create the ACR:

```
az acr create --name <ACR_NAME> --resource-group <RESOURCE_GROUP_NAME> --sku
↪Basic --output table
```

where:

- <ACR\_NAME> must be between 5-50 alphanumeric characters and is unique to Azure. If you're not sure your chosen name is available, you can run `az acr check-name --name <ACR_NAME> --output table`.
- --sku is the pricing and capacity tier for the registry. See [this page](#) for more details.

#### 5. Login in the ACR:

```
az acr login --name <ACR_NAME>
```

#### 6. Note down the AppID of the ACR:

```
az acr show --name <ACR_NAME> --query "id" -o tsv
```

We need this in order to assign the AcrPush role which will allow BinderHub to push images to the registry. You can save this to a bash variable like so:

```
ACR_ID=$(az acr show --name <ACR_NAME> --query "id" -o tsv)
```

#### 7. Create a Service Principal with the AcrPush role assignment:

```
az ad sp create-for-rbac --name <SP_NAME> --role AcrPush --scope <ACR_ID>
```

where:

- <SP\_NAME> is a recognisable name for your Service Principal, for example `binderhub-sp`,
- <ACR\_ID> is the AppID we retrieved in step 6 above. You can replace this with `${ACR_ID}` if you saved it to a bash variable.

**Important:** Note down the AppID and password that are output by this step. These are the login credentials BinderHub will use to access the registry.

**The password will not be recoverable after this step, so make sure you keep it safe!**

If you'd like to save this info to bash variables, you can replace step 8 with the following commands:

```
SERVICE_PRINCIPAL_PASSWORD=$(az ad sp create-for-rbac --name <SP_NAME> --role AcrPush
↪--scopes <ACR_ID> --query password --output tsv)
SERVICE_PRINCIPAL_ID=$(az ad sp show --id http://<SP_NAME> --query appId --output tsv)
```

## Next step

Now that our cloud resources are set up, it's time to *Set up BinderHub*.

### 3.1.3 Set up BinderHub

BinderHub uses Helm Charts to set up the applications we'll use in our Binder deployment. If you're curious about what Helm Charts are and how they're used here, see the [Zero to JupyterHub guide](#).

Below we'll cover how to configure your Helm Chart, and how to create your BinderHub deployment.

#### Preparing to install

To configure the Helm Chart we'll need to generate several pieces of information and insert them into `yaml` files.

First we'll create a folder where we'll store our BinderHub configuration files. You can do so with the following commands:

```
mkdir binderhub
cd binderhub
```

Now we'll collect the information we need to deploy our BinderHub. The first is the content of the JSON file created when we set up the container registry. For more information on getting a registry password, see [Set up the container registry](#). We'll copy/paste the contents of this file in the steps below.

Create two random tokens by running the following commands then copying the outputs.:

```
openssl rand -hex 32
openssl rand -hex 32
```

---

**Note:** This command is run **twice** because we need two different tokens.

---

#### Create `secret.yaml` file

Create a file called `secret.yaml` and add the following:

```
jupyterhub:
  hub:
    services:
      binder:
        apiToken: "<output of FIRST `openssl rand -hex 32` command>"
  proxy:
    secretToken: "<output of SECOND `openssl rand -hex 32` command>"
```

Next, we'll configure this file to connect with our registry.

#### If you are using `gcr.io`

Add the information needed to connect with the registry to `secret.yaml`. You'll need the content in the JSON file that was created when we created our `gcr.io` registry account. Below we show the structure of the YAML you need to insert. Note that the first line is not indented at all:

```
registry:
  url: https://gcr.io
  # below is the content of the JSON file downloaded earlier for the container_
  ↪registry from Service Accounts
```

(continues on next page)



(continued from previous page)

```
# it will look something like the following (with actual values instead of empty_
↪strings)
# paste the content after `password: |` below
password: |
{
  "type": "<REPLACE>",
  "project_id": "<REPLACE>",
  "private_key_id": "<REPLACE>",
  "private_key": "<REPLACE>",
  "client_email": "<REPLACE>",
  "client_id": "<REPLACE>",
  "auth_uri": "<REPLACE>",
  "token_uri": "<REPLACE>",
  "auth_provider_x509_cert_url": "<REPLACE>",
  "client_x509_cert_url": "<REPLACE>"
}
```

**Tip:**

- The content you put just after password: | must all line up at the same tab level.
- Don't forget the | after the password: label.

**If you are using Docker Hub**

Update secret.yaml by entering the following:

```
registry:
  username: <docker-id>
  password: <password>
```

**Note:**

- <docker-id> and <password> are your credentials to login to Docker Hub. If you use an organization to store your Docker images, this account must be a member of it.

**If you are using Azure Container Registry**

Update secret.yaml to include the following:

```
registry:
  url: https://<ACR_NAME>.azurecr.io
  username: <SERVICE_PRINCIPAL_ID>
  password: <SERVICE_PRINCIPAL_PASSWORD>
```

where:

- <ACR\_NAME> is the name you gave to your Azure Container Registry,
- <SERVICE\_PRINCIPAL\_ID> is the AppID of the Service Principal with AcrPush role assignment,
- <SERVICE\_PRINCIPAL\_PASSWORD> is the password for the Service Principal.

## Create config.yaml

Create a file called `config.yaml` and choose the following directions based on the registry you are using.

### If you are using gcr.io

To configure BinderHub to use `gcr.io`, simply add the following to your `config.yaml` file:

```
config:
  BinderHub:
    use_registry: true
    image_prefix: gcr.io/<google-project-id>/<prefix>-
```

---

#### Note:

- `<google-project-id>` can be found in the JSON file that you pasted above. It is the text that is in the `project_id` field. This is the project *ID*, which may be different from the project *name*.
  - `<prefix>` can be any string, and will be prepended to image names. We recommend something descriptive such as `binder-dev-` or `binder-prod-` (ending with a `-` is useful).
  - Note that in both cases, you should remove the `<` and `>` symbols, they are simply placeholders in the code above.
- 

### If you are using Docker Hub

Update `config.yaml` by entering the following:

```
config:
  BinderHub:
    use_registry: true
    image_prefix: <docker-id OR organization-name>/<prefix>-
```

---

#### Note:

- `“<docker-id OR organization-name>“` is where you want to store Docker images. This can be your Docker ID account **or** an organization that your account belongs to.
  - `“<prefix>“` can be any string, and will be prepended to image names. We recommend something descriptive such as `binder-dev-` or `binder-prod-` (ending with a `-` is useful).
- 

### If you are using Azure Container Registry

If you want your BinderHub to push and pull images from an Azure Container Registry (ACR), then your `config.yaml` file will look as follows:

```
config:
  BinderHub:
    use_registry: true
    image_prefix: <ACR_NAME>.azurecr.io/<project-name>/<prefix>-
  DockerRegistry:
    token_url: "https://<ACR_NAME>.azurecr.io/oauth2/token?service=<ACR_NAME>.azurecr.
→io"
```

(continues on next page)

(continued from previous page)

where:

- `<ACR_NAME>` is the name you gave to your ACR,
- `<project-name>` is an arbitrary name that is required due to BinderHub assuming that `image_prefix` contains an extra level for the project name. See [this issue](#) for further discussion. If this is not provided, you may find BinderHub rebuilds images every launch instead of pulling them from the ACR. Suggestions for `<project-name>` could be `ACR_NAME` or the name you give your BinderHub.

### If you are using a custom registry

Authenticating with a Docker registry is slightly more complicated. BinderHub knows how to talk to gcr.io and DockerHub, but if you are using another registry, you will have to provide more information, in the form of two different urls:

- registry url (added to `docker/config.json`)
- token url for authenticating access to the registry

First, setup the docker configuration with the host used for authentication:

```
registry:
  url: "https://myregistry.io"
  username: xxx
  password: yyy
```

This creates a docker config.json used to check for images in the registry and push builds to it.

Second, you will need to instruct BinderHub about the token URL:

```
config:
  BinderHub:
    use_registry: true
    image_prefix: "your-registry.io/<prefix>-"
  DockerRegistry:
    token_url: "https://myregistry.io/v2/token?service="
```

If you setup your own local registry using [native basic HTTP authentication](#) (htpasswd), you can set `token_url` to None.

**Note:** There is one additional URL to set in the unlikely event that docker config.json must use a different URL to refer to a registry than the registry's actual url. If this is the case, `registry.url` at the top-level must match `DockerRegistry.auth_config_url`:

```
registry:
  url: "https://"
```

It's not clear that this can ever be the case for custom registries, however it is the case for DockerHub:

```
registry:
  url: "https://index.docker.io/v1"
config:
  DockerRegistry:
    url: "https://registry.hub.docker.com" # the actual v2 registry url
```

(continues on next page)

(continued from previous page)

```
auth_config_url: "https://index.docker.io/v1" # must match above!
token_url: "https://auth.docker.io/token?service=registry.docker.io"
```

however, BinderHub is aware of DockerHub's peculiarities and can handle these without any additional configuration beyond `registry.url`.

---

**Important:** BinderHub assumes that `image_prefix` contains an extra level for the project name such that: `gcr.io/<project-id>/<prefix>-name:tag`. Hence, your `image_prefix` field should be set to: `your-registry.io/<some-project-name>/<prefix>-`. See [this issue](#) for more details.

`<some-project-name>` can be completely arbitrary and/or made-up. For example, it could be the name you give your BinderHub. Without this extra level, you may find that your BinderHub always rebuilds images instead of pulling them from the registry.

---

## Install BinderHub

First, get the latest helm chart for BinderHub.:

```
helm repo add jupyterhub https://jupyterhub.github.io/helm-chart
helm repo update
```

Next, **install the Helm Chart** using the configuration files that you've just created. Do this by running the following command:

```
helm install jupyterhub/binderhub --version=0.2.0-3b53fce --name=<choose-name> --
↪namespace=<choose-namespace> -f secret.yaml -f config.yaml
```

This command will install the Helm chart released on March 3rd, 2019 as identified by the commit hash (the random string after `0.2.0-`), which is provided as a working example. You should provide the commit hash for the most recent release, which can be found [here](#).

---

### Note:

- `--version` refers to the version of the BinderHub **Helm Chart**. Available versions can be found [here](#).
  - `name` and `namespace` may be different, but we recommend using the same `name` and `namespace` to avoid confusion. We recommend something descriptive and short, such as `binder`.
  - If you run `kubectl get pod --namespace=<namespace-from-above>` you may notice the `binder` pod in `CrashLoopBackoff`. This is expected, and will be resolved in the next section.
- 

This installation step will deploy both a BinderHub and a JupyterHub, but they are not yet set up to communicate with each other. We'll fix this in the next step. Wait a few moments before moving on as the resources may take a few minutes to be set up.

## Connect BinderHub and JupyterHub


In the google console, run the following command to print the IP address of the JupyterHub we just deployed.:

```
kubectl --namespace=<namespace-from-above> get svc proxy-public
```

Copy the IP address under `EXTERNAL-IP`. This is the IP of your JupyterHub. Now, add the following lines to `config.yaml` file:

```
config:
  BinderHub:
    hub_url: http://<IP in EXTERNAL-IP>
```

Next, upgrade the helm chart to deploy this change:

```
helm upgrade <name-from-above> jupyterhub/binderhub --version=0.2.0-3b53fce -f secret.yaml -f config.yaml
```

For the first deployment of your BinderHub, the commit hash parsed to the `--version` argument should be the same as in step 3.4. However, when it comes to updating your BinderHub, you can parse the commit hash of a newer chart version.

## Try out your BinderHub Deployment

If the `helm upgrade` command above succeeds, it's time to try out your BinderHub deployment.

First, find the IP address of the BinderHub deployment by running the following command:

```
kubectl --namespace=<namespace-from-above> get svc binder
```

Note the IP address in `EXTERNAL-IP`. This is your BinderHub IP address. Type this IP address in your browser and a BinderHub should be waiting there for you.

You now have a functioning BinderHub at the above IP address.

## Customizing your Deployment

The Helm chart used to install your BinderHub deployment exposes a number of optional features. Below we describe a few of the most common customizations and how you can configure them.

### Increase your GitHub API limit

**Note:** Increasing the GitHub API limit is not strictly required, but is recommended before sharing your BinderHub URL with users.

By default GitHub only lets you make 60 requests each hour. If you expect your users to serve repositories hosted on GitHub, we recommend creating an API access token to raise your API limit to 5000 requests an hour.

1. Create a new token with default (check no boxes) permissions [here](#).
2. Store your new token somewhere secure (e.g. keychain, netrc, etc.)
3. Update `secret.yaml` by entering the following:

```
config:
  GitHubRepoProvider:
    access_token: <insert_token_value_here>
```

BinderHub will use this token when making API requests to GitHub. See the [GitHub authentication documentation](#) for more information about API limits.

## Accessing private repositories

By default, BinderHub doesn't have access to private repositories (repositories that require credentials to clone). Since users never enter credentials into BinderHub, BinderHub *itself* must be given permission to clone any private repositories you want BinderHub to be able to build.

**Warning:** Since cloning is done 'as binderhub' this means that any user can build any private repository that BinderHub has access to. They may be private from the wider world, but they are not private from other users with access to the same BinderHub.

## GitHub

Granting permission follows the the same steps above in [Customizing your Deployment](#) to create a GitHub access token and configure BinderHub to use it. Previously, the token only needed minimal read-only permissions (the default). In order to access private repositories, the token must have **full read/write permissions on all your repos**<sup>1</sup>.

## New personal access token

---

Personal access tokens function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

### Token description

BinderHub token

What's this token for?

### Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

<input checked="" type="checkbox"/>	<b>repo</b>	Full control of private repositories
<input checked="" type="checkbox"/>	repo:status	Access commit status
<input checked="" type="checkbox"/>	repo_deployment	Access deployment status
<input checked="" type="checkbox"/>	public_repo	Access public repositories
<input checked="" type="checkbox"/>	repo:invite	Access repository invitations

Fig. 1: Creating a token with the full `repo` scope needed in order to clone private repos.

You can set these permissions when you create the token, or change them after the fact by editing the token's permissions at any time at [the token administration page](#).

## GitLab

To access private GitLab repos, create an API token for your binderhub user under "User Settings" > "Access tokens". It at least needs the scopes "api" and "read\_repository".

Then update `secret.yaml` with the following:

---

<sup>1</sup> Hopefully in the future, GitHub will allow more granular permissions for private repos.

User Settings &gt; Access Tokens

## Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API.

You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

### Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

#### Name

#### Expires at

#### Scopes

☒ **api**

Grants complete read/write access to the API, including all groups and projects.

☐ **read\_user**

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

☒ **read\_repository**

Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

☐ **read\_registry**

Grants read-only access to container registry images on private projects.

[Create personal access token](#)

```
config:
  GitLabRepoProvider:
    private_token: <insert_token_value_here>
```

This token will be used for accessing the GitLab API, and is also used as the Git password when cloning repos. With this token, no username is required to clone a repo.

## Use Docker-inside-Docker (DinD)

By default, BinderHub will build pods with the host Docker installation. This often means you are stuck with whatever version of Docker provided by your cloud provider. BinderHub supports an alternative that uses [Docker-in-Docker \(DinD\)](#). To turn dind on, you'll need to set the following configuration in your `config.yaml` file:

```
dind:
  enabled: true
  daemonset:
    image:
      name: docker
      tag: 18.09.2-dind
```

If you plan to host multiple BinderHub deployments on the same kubernetes cluster, you'll also need to isolate the host socket and library directory for each DinD application:

```
dind:
  hostLibDir: /var/lib/dind/"<name of deployment, e.g. staging>"
```

(continues on next page)

(continued from previous page)

```
hostSocketDir: /var/run/dind/"<name of deployment, e.g. staging>"
```

For next steps, see *Debugging BinderHub* and *Tear down your Binder deployment*.

### 3.1.4 Tear down your Binder deployment

Deconstructing a Binder deployment can be a little bit confusing because users may have caused new cloud containers to be created. It is important to remember to delete each of these containers or else they will continue to exist (and cost money!).

#### Contracting the size of your cluster

If you would like to shrink the size of your cluster, refer to the [Expanding and contracting the size of your cluster](#) section of the [Zero to JupyterHub](#) documentation. Resizing the cluster to zero nodes could be used if you wish to temporarily reduce the cluster (and save costs) without deleting the cluster.

#### Deleting the cluster

To delete a Binder cluster, follow the instructions in the [Turning Off JupyterHub and Computational Resources](#) section of the [Zero to JupyterHub](#) documentation.

---

**Important:** Double check your cloud provider account to make sure all resources have been deleted as expected. Double checking is a good practice and will help prevent unwanted charges.

---



---

## Customization and deployment information

---

Information on how to customize your BinderHub as well as explore what others in the community have done.

### 4.1 Customization

Information on how to customize your BinderHub as well as explore what others in the community have done.

#### 4.1.1 Debugging BinderHub

If BinderHub isn't behaving as you'd expect, you'll need to debug your kubernetes deployment of the JupyterHub and BinderHub services. For a guide on how to debug in Kubernetes, see the [Zero to JupyterHub debugging guide](#).

#### Indentation in Config Files

Probably the most common cause of unexplained behaviour from a BinderHub is an incorrectly indented key in one of the configuration files. Indentation levels can change depending on how the BinderHub was deployed. Or in other words, the structure of the values is dictated by the chart being configured.

If you deployed BinderHub by following [Zero to BinderHub](#), then you probably have `secret.yaml` and `config.yaml` files. This is the case where one chart includes another. Here, the BinderHub chart includes the JupyterHub chart. The included chart values are loaded from a key with the chart's name, within the parent `config.yaml` file.:

```
# BinderHub chart config is top-level
registry:
  username: "a-username"
  password: "xxxx"

# JupyterHub chart config is now under "jupyterhub" key
# This key is dictated by the name of the included chart
jupyterhub:
  hub:
```

(continues on next page)

(continued from previous page)

```
resources:
  memory:
    limit: '1G'
```

If you were deploying only a JupyterHub, it only has one chart and so many of the Hub-related keys become top-level, such as “hub”, “singleuser”, “auth” and so on.:

```
auth:
  type: "github"

hub:
  resources:
    memory:
      limit: "1G"
```

If you are using the [JupyterHub for Kubernetes Customization Guide](#), then this is an important difference to note. Any keys you add **to the BinderHub** from this guide, should go under the `jupyterhub` key in `config.yaml` file as they are specific to the JupyterHub chart included within BinderHub.

For completeness, another case is how [mybinder.org](#) itself is deployed. The BinderHub serving [mybinder.org](#) is deployed as a dependency of a local chart and so `binderhub` becomes the top-level key with `jupyterhub` nested below.:

```
binderhub:
  registry:
    ...

  jupyterhub:
    hub:
      ...
```

Such kinds of “nested” chart dependencies are managed by a special file called `requirements.yaml`. More info on using such a file can be found in the [related Helm docs](#).

Ok so now that we’ve ascertained that indentation errors are the most likely cause of undesirable behaviour, how can we prevent them?

One tool that can be used to combat this is [helm diff](#) which shows what would change in the chart between upgrades, releases, etc. If you think something should change but the diff is empty, that’s probably a good clue something is misplaced!

An automated script to check linting may also be of appeal. The Jupyter Team have a [lint script](#) to check the validity of the JupyterHub chart. It runs `yamllint`, `helm lint` and `kubeval`.

For further discussion on this topic or to join the conversation, you can visit [this Discourse thread](#) or [this GitHub issue](#).

## Changing the helm chart

If you make changes to your Helm Chart (e.g., while debugging), you should run an upgrade on your Kubernetes deployment like so:

```
helm upgrade binder jupyterhub/binderhub --version=<commit-hash> -f secret.yaml -f_
↪config.yaml
```

where `<commit-hash>` can be found [here](#).

## 4.1.2 Customizing your BinderHub deployment

### JupyterHub customization

Because BinderHub uses JupyterHub to manage all user sessions, you can customize many aspects of the resources available to the user. This is primarily done by modifications to your BinderHub's Helm chart (`config.yaml`).

To make edits to your JupyterHub deployment via `config.yaml`, use the following pattern:

```
binderhub:
  jupyterhub:
    <JUPYTERHUB-CONFIG-YAML>
```

For example, see [this section of the mybinder.org Helm Chart](#).

For information on how to configure your JupyterHub deployment, see the [JupyterHub for Kubernetes Customization Guide](#).

### About page customization

BinderHub serves a simple about page at `https://BINDERHOST/about`. By default this shows the version of BinderHub you are running. You can add additional HTML to the page by setting the `c.BinderHub.about_message` configuration option to the raw HTML you would like to add. You can use this to display contact information or other details about your deployment.

### Template customization

BinderHub uses [Jinja](#) template engine and it is possible to customize templates in a BinderHub deployment. Here it is explained by a minimal example which shows how to use a custom logo.

Before configuring BinderHub to use custom templates and static files, you have to provide these files to the binder pod where the application runs. One way to do this using [Init Containers](#) and a Git repo.

Firstly assume that you have a Git repo `binderhub_custom_files` which holds your custom files:

```
binderhub_custom_files/
├── static
│   └── custom_logo.svg
└── templates
    └── page.html
```

where `page.html` extends the [base page.html](#) and updates only the source url of the logo in order to use your custom logo:

```
{% extends "templates/page.html" %}

{% block logo_image %}"{{ EXTRA_STATIC_URL_PREFIX }}custom_logo.svg"{% endblock logo_
↪image %}
```

---

**Note:** If you want to extend any other base template, you have to include `{% extends "templates/<base_template_name>.html" %}` in the beginning of your custom template. It is also possible to have completely new template instead of extending the base one. Then BinderHub will ignore the base one.

---

Now you can use `Init Containers` to clone that Git repo into a volume (`custom-templates`) which is mounted to both `init` container and `binder` container. To do that add the following into your `config.yaml`:

```
initContainers:
- name: git-clone-templates
  image: alpine/git
  args:
    - clone
    - --single-branch
    - --branch=master
    - --depth=1
    - --
    - <repo_url>
    - /etc/binderhub/custom
  securityContext:
    runAsUser: 0
  volumeMounts:
    - name: custom-templates
      mountPath: /etc/binderhub/custom
extraVolumes:
- name: custom-templates
  emptyDir: {}
extraVolumeMounts:
- name: custom-templates
  mountPath: /etc/binderhub/custom
```

---

**Note:** You have to replace `<repo_url>` with the url of the public repo (binderhub\_custom\_files) where you have your templates and static files.

---

The final thing you have to do is to configure BinderHub, so it knows where to look for custom templates and static files (where the volume is mounted). To do that update your `config.yaml` by the following:

```
config:
  BinderHub:
    template_path: /etc/binderhub/custom/templates
    extra_static_path: /etc/binderhub/custom/static
    extra_static_url_prefix: /extra_static/
    template_variables:
      EXTRA_STATIC_URL_PREFIX: "/extra_static/"
```

**Warning:** You have to set the `extra_static_url_prefix` different than `/static/` which is the default static url prefix of BinderHub. Otherwise default one overrides it and BinderHub only uses default static files.

---

**Note:** In this example a custom template variable (`EXTRA_STATIC_URL_PREFIX`) to hold the value of `extra_static_url_prefix` is also defined, which was used in `custom page.html`. This is good to do specially if you have many custom templates and static files.

---

## Custom configuration for specific repositories

Sometimes you would like to provide a repository-specific configuration. For example, if you'd like certain repositories to have **higher pod quotas** than others, or if you'd like to provide certain resources to a subset of repositories.

To override the configuration for a specific repository, you can provide a list of dictionaries that allow you to provide

a pattern to match against each repository's specification, and override configuration values for any repositories that match this pattern.

**Note:** If you provide **multiple patterns that match a single repository** in your spec-specific configuration, then **later values in the list will override earlier values**.

To define this list of patterns and configuration overrides, use the following pattern in your Helm Chart (here we show an example using `GitHubRepoProvider`, but this works for other `RepoProviders` as well):

```
config:
  GitHubRepoProvider:
    spec_config:
      - pattern: ^ines/spacy-binder.*:
        config:
          key1: value1
      - pattern: pattern2
        config:
          key1: othervalue1
          key2: othervalue2
```

For example, the following specification configuration will assign a pod quota of 999 to the spacy-binder repository, and a pod quota of 1337 to any repository in the JupyterHub organization.

```
config:
  GitHubRepoProvider:
    spec_config:
      - pattern: ^ines/spacy-binder.*:
        config:
          quota: 999
      - pattern: ^jupyterhub.*
        config:
          quota: 1337
```

### 4.1.3 Enabling Authentication

By default BinderHub runs without authentication and for each launch it creates a temporary user and starts a server for that user.

In order to enable authentication for BinderHub by using JupyterHub as an oauth provider, you need to add the following into `config.yaml`:

```
config:
  BinderHub:
    auth_enabled: true

jupyterhub:
  cull:
    # don't cull authenticated users
    users: False

  hub:
    redirectToServer: false
    services:
      binder:
        oauth_redirect_uri: "http://<binderhub_url>/oauth_callback"
```

(continues on next page)

(continued from previous page)

```

    oauth_client_id: "binder-oauth-client-test"
extraConfig:
  binder: |
    from kubespawner import KubeSpawner

    class BinderSpawner(KubeSpawner):
        def start(self):
            if 'image' in self.user_options:
                # binder service sets the image spec via user options
                self.image = self.user_options['image']
            return super().start()
    c.JupyterHub.spawner_class = BinderSpawner

singleuser:
  # to make notebook servers aware of hub
  cmd: jupyterhub-singleuser

auth: {}

```

If the configuration above was entered correctly, once you upgrade your BinderHub Helm Chart with `helm upgrade...`, users that arrive at your BinderHub URL will be directed to a login page. Once they enter their credentials, they'll be taken to the typical BinderHub landing page.

**Note:** If users *don't* go to a BinderHub landing page after they log-in, then the configuration above is probably incorrect. Double-check that the BinderHub configuration (and the JupyterHub authentication configuration) look good.

**Note:** For `jupyterhub.auth` you should use config of your authenticator. For more information you can check [the Authentication guide](#).

**Warning:** `jupyterhub-singleuser` requires JupyterHub to be installed in user server images. Therefore ensure that you use at least `jupyter/repo2docker:ccce3fe` image to build user images. Because `repo2docker` installs JupyterHub by default after that.

## Authentication with named servers

With above configuration Binderhub limits each authenticated user to start one server at a time. When a user already has a running server, BinderHub displays an error message.

If you want to have users be able to launch multiple servers at the same time, you have to enable named servers on JupyterHub:

```

config:
  BinderHub:
    use_named_servers: true
  jupyterhub:
    hub:
      allowNamedServers: true
      # change this value as you wish,

```

(continues on next page)

(continued from previous page)

```
# or remove this line if you don't want to have any limit
namedServerLimitPerUser: 5
```

**Note:** BinderHub assigns a unique name to each server with max 40 characters.

#### 4.1.4 Secure with HTTPS

To enable HTTPS on your BinderHub you can setup an ingress proxy and configure it to serve both, the Binder and JupyterHub interface, using TLS. You can either manually provide TLS certificates or use [Let's Encrypt](#) to automatically get signed certificates.

##### Setup IP & domain

1. Get a static IP(v4) address that you will assign to your ingress proxy later. For example, on Google Cloud this can be done using `gcloud compute addresses create <alias-name-for-ip> --region <region>` and retrieve the assigned IP using `gcloud compute addresses list`.
2. Buy a domain name from a registrar. Pick whichever one you want.
3. Set A records to your above retrieved external IP, one for Binder and one for JupyterHub. We need two distinct subdomains for the routing to the two different services as they will be served by the same ingress proxy. We suggest you use `hub.binder.` for JupyterHub and `binder.` for your BinderHub. Once you are done your BinderHub will be available at `https://binder..`
4. Wait some minutes for the DNS A records to propagate.

##### cert-manager for automatic TLS certificate provisioning

To automatically generate TLS certificates and sign them using [Let's Encrypt](#), we utilise `cert-manager`. Installation is done by using the following command:

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v0.11.0/
↪cert-manager.yaml
```

For installations of Kubernetes v1.15 or below, you also need to supply `--validate=false` to the above command. For more detail on this, see the [Getting Started guide](#).

We then need to create an issuer that will contact Let's Encrypt for signing our certificates. Use the following template to create a new file `binderhub-issuer.yaml` and instantiate it using `kubectl apply -f binderhub-issuer.yaml`.

```
apiVersion: cert-manager.io/v1alpha2
kind: Issuer
metadata:
  name: letsencrypt-production
  namespace: <same-namespace-as-binderhub>
spec:
  acme:
    # You must replace this email address with your own.
    # Let's Encrypt will use this to contact you about expiring
    # certificates, and issues related to your account.
```

(continues on next page)

(continued from previous page)

```
email: <your-contact-mail>
server: https://acme-v02.api.letsencrypt.org/directory
privateKeySecretRef:
  # Secret resource used to store the account's private key.
  name: letsencrypt-production
solvers:
- http01:
    ingress:
      class: nginx
```

See the documentation for [more details on configuring the issuer](#).

## Ingress proxy using nginx

We will use the [nginx ingress controller](#) to proxy the TLS connection to our BinderHub setup. This will run using the static IP we have acquired before. We therefore create a new configuration file `nginx-ingress.yaml`:

```
controller:
  service:
    loadBalancerIP: <STATIC-IP>
```

Afterwards we install the ingress proxy using `helm install stable/nginx-ingress --name binderhub-proxy --namespace <same-namespace-as-binderhub> -f nginx-ingress.yaml`. Then wait until it is ready and showing the correct IP when looking at the output of `kubectl --namespace <same-namespace-as-binderhub> get services binderhub-proxy-nginx-ingress-controller`.

## Adjust BinderHub config to serve via HTTPS

With the static IP, DNS records and ingress proxy setup, we can now change our BinderHub configuration to serve traffic via HTTPS. Therefore adjust your `config.yaml` with the following sections and apply it using `helm upgrade ....`

```
config:
  BinderHub:
    hub_url: https://<jupyterhub-URL>
  jupyterhub:
    ingress:
      enabled: true
      hosts:
        - <jupyterhub-URL>
      annotations:
        kubernetes.io/ingress.class: nginx
        kubernetes.io/tls-acme: "true"
        cert-manager.io/issuer: letsencrypt-production
      https:
        enabled: true
        type: nginx
    tls:
      - secretName: <jupyterhub-URL-with-dashes-instead-of-dots>-tls
        hosts:
          - <jupyterhub-URL>
```

(continues on next page)



(continued from previous page)

```
ingress:
  enabled: true
  hosts:
    - <binderhub-URL>
  annotations:
    kubernetes.io/ingress.class: nginx
    kubernetes.io/tls-acme: "true"
    cert-manager.io/issuer: letsencrypt-production
  https:
    enabled: true
    type: nginx
  tls:
    - secretName: <binderhub-URL-with-dashes-instead-of-dots>-tls
      hosts:
        - <binderhub-URL>
```

Once the `helm upgrade ...` command has been run, it may take up to 10 minutes until the certificates are issued. You can check their status using `kubectl describe certificate --namespace <binderhub-namespace> <binderhub-URL>-tls`.



---

## BinderHub Developer and Architecture Documentation

---

A more detailed overview of the BinderHub design, architecture, and functionality.

### 5.1 Administrators and developers

A more detailed overview of the BinderHub design, architecture, and functionality.

#### 5.1.1 The BinderHub Architecture

This page provides a high-level overview of the technical pieces that make up a BinderHub deployment.

##### Tools used by BinderHub

BinderHub connects several services together to provide on-the-fly creation and registry of Docker images. It utilizes the following tools:

- A **cloud provider** such Google Cloud, Microsoft Azure, Amazon EC2, and others
- **Kubernetes** to manage resources on the cloud
- **Helm** to configure and control Kubernetes
- **Docker** to use containers that standardize computing environments
- A **BinderHub UI** that users can access to specify Git repos they want built
- **BinderHub** to generate Docker images using the URL of a Git repository
- A **Docker registry** (such as gcr.io) that hosts container images
- **JupyterHub** to deploy temporary containers for users

## What happens when a user clicks a Binder link?

After a user clicks a Binder link, the following chain of events happens:

1. BinderHub resolves the link to the repository.
2. BinderHub determines whether a Docker image already exists for the repository at the latest `ref` (git commit hash, branch, or tag).
3. **If the image doesn't exist**, BinderHub creates a `build pod` that uses `repo2docker` to do the following:
  - Fetch the repository associated with the link
  - Build a Docker container image containing the environment specified in `configuration files` in the repository.
  - Push that image to a Docker registry, and send the registry information to the BinderHub for future reference.
4. BinderHub sends the Docker image registry to **JupyterHub**.
5. JupyterHub creates a Kubernetes pod for the user that serves the built Docker image for the repository.
6. JupyterHub monitors the user's pod for activity, and destroys it after a short period of inactivity.

## A diagram of the BinderHub architecture

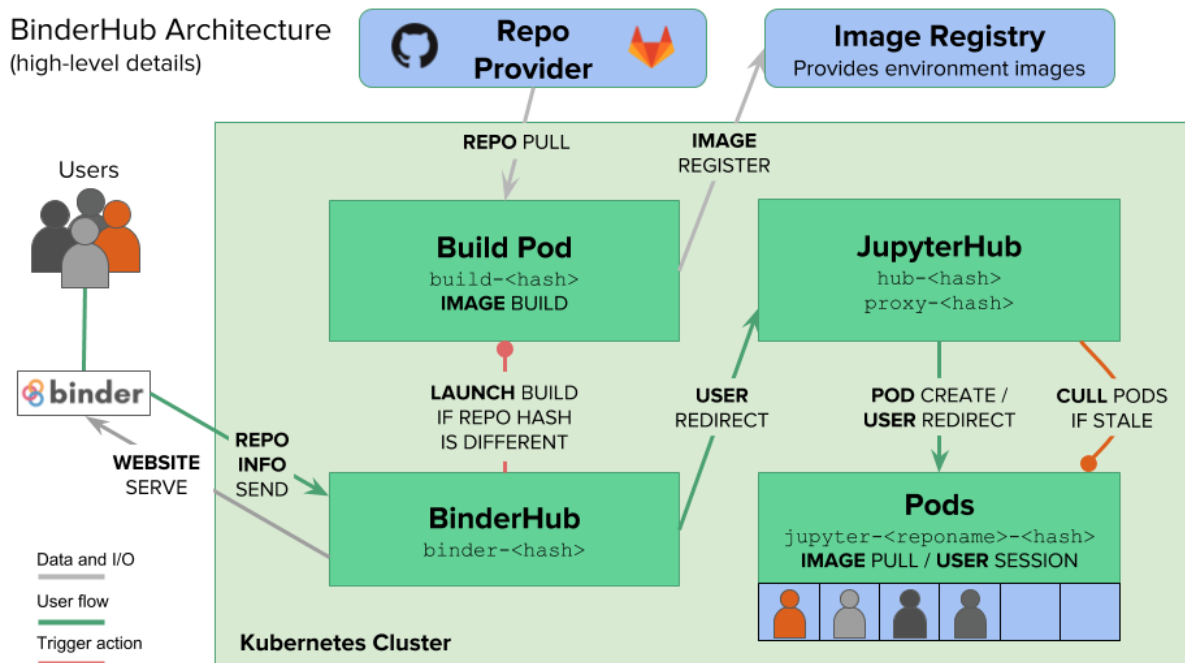


Fig. 1: Here is a high-level overview of the components that make up BinderHub.

### 5.1.2 Event Logging

Events are discrete & structured items emitted by BinderHub when specific events happen. For example, the `binderhub.jupyter.org/launch` event is emitted whenever a Launch succeeds.

These events may be sent to a *sink* via handlers from the `python logging` module.

## Events vs Metrics

BinderHub also exposes [prometheus](#) metrics. These are pre-aggregated, and extremely limited in scope. They can efficiently answer questions like ‘how many launches happened in the last hour?’ but not questions like ‘how many times was this repo launched in the last 6 months?’. Events are discrete and can be aggregated in many ways during analysis. Metrics are aggregated at source, and this limits what can be done with them during analysis. Metrics are mostly operational, while events are for analytics.

## What events to emit?

Since events have a lot more information than metrics do, we should be careful about what events we emit. In general, we should pose an **explicit question** that events can answer.

For example, to answer the question *How many times has my GitHub repo been launched in the last 6 months?*, we would need to emit an event every time a launch succeeds. To answer the question *how long did users spend on my repo?*, we would need to emit an event every time a user notebook is killed, along with the lifetime length of the notebook.

[Wikimedia’s EventLogging Guidelines](#) contain a lot of useful info on how to approach adding more events.

## BinderHub Events

### Launch event

This event is emitted whenever a new repo is launched.

Schemas:

- [version 1](#)
- [version 2](#)
- [version 3](#)

## 5.1.3 BinderHub API Documentation

### Endpoint

There’s one API endpoint, which is:

```
/build/<provider_prefix>/<spec>
```

Even though it says **build** it actually performs **launch**.

**provider\_prefix** identifies the provider. **spec** defines the source of the computing environment to be built and served using the given provider. See [Providers](#) for supported inputs.

## Providers

Currently supported providers, their prefixes and specs are:

Provider	provider_prefix	spec	notes
GitHub	gh	<user>/<repo>/<commit-sha-or-tag-or-branch>	
Git	git	<url-escaped-url>/<commit-sha>	arbitrary HTTP git repos
GitLab	gl	<url-escaped-namespace>/<commit-sha-or-tag-or-branch>	
Gist	gist	<github-username>/<gist-id><commit-sha-or-tag>	

Next, construct an appropriate URL and send a request.

You'll get back an [Event Stream](#). It's pretty much just a long-lived HTTP connection with a well known JSON based data protocol. It's one-way communication only (server to client) and is straightforward to implement across multiple languages.

When the request is received, the following happens:

1. Check if this image exists in our cached image registry. If so, launch it.
2. If it doesn't exist in the image registry, we check if a build is currently running. If it is, we attach to it and start streaming logs from it to the user.
3. If there is no build in progress, we start a build and start streaming logs from it to the user.
4. If the build succeeds, we contact the JupyterHub API and start launching the server.

## Events

This section catalogs the different events you might receive.

### Failed

Emitted whenever a build or launch fails. You *must* close your EventStream when you receive this event.

```
{'phase': 'failed', 'message': 'Reason for failure'}
```

### Built

Emitted after the image has been built, before launching begins. This is emitted in the start if the image has been found in the cache registry, or after build completes successfully if we had to do a build.

```
{'phase': 'built', 'message': 'Human readable message', 'imageName': 'Full name of_  
↪the image that is in the cached docker registry'}
```

Note that clients shouldn't rely on the imageName field for anything specific. It should be considered an internal implementation detail.

## Waiting

Emitted when we started a build pod and are waiting for it to start.

```
{'phase': 'waiting', 'message': 'Human readable message'}
```

## Building

Emitted during the actual building process. Direct stream of logs from the build pod from repo2docker, in the same form as logs from a normal docker build.

```
{'phase': 'building', 'message': 'Log message'}
```

## Fetching

Emitted when fetching the repository to be built from its source (GitHub, GitLab, wherever).

```
{'phase': 'fetching', 'message': 'log messages from fetching process'}
```

## Pushing

Emitted when the image is being pushed to the cache registry. This provides structured status info that could be in a progressbar. It's structured similar to the output of docker push.

```
{'phase': 'pushing', 'message': 'Human readable message', 'progress': {'layer1': {
  ↳ 'current': <bytes-pushed>, 'total': <full-bytes>}, 'layer2': {'current': <bytes-
  ↳ pushed>, 'total': <full-bytes>}, 'layer3': "Pushed", 'layer4': 'Layer already exists
  ↳ '}}
```

## Launching

When the repo has been built, and we're in the process of waiting for the hub to launch. This could end up succeeding and emitting a 'ready' event or failing and emitting a 'failed' event.

```
{'phase': 'launching', 'message': 'user friendly message'}
```

## Ready

When your notebook is ready! You get a endpoint URL and a token used to access it. You can access the notebook / API by using the token in one of the ways the [notebook accepts security tokens](#).

```
{ "phase": "ready", "message": "Human readable message", "url": "full-url-of-notebook-
  ↳ server", "token": "notebook-server-token" }
```

## Heartbeat

In EventSource, all lines beginning with : are considered comments. We send a :heartbeat every 30s to make sure that we can pass through proxies without our request being killed.

## 5.1.4 Configuration and Source Code Reference

### app

Module: `binderhub.app`

The binderhub application

### BinderHub

```
class binderhub.app.BinderHub(**kwargs)
```

An Application for starting a builder.

```
config c.BinderHub.about_message = Unicode('')
```

Additional message to display on the about page.

Will be directly inserted into the about page's source so you can use raw HTML.

```
config c.BinderHub.appendix = Unicode('')
```

Appendix to pass to repo2docker

A multi-line string of Docker directives to run. Since the build context cannot be affected, ADD will typically not be useful.

This should be a Python string template. It will be formatted with at least the following names available:

- `binder_url`: the shareable URL for the current image (e.g. for sharing links to the current Binder)
- `repo_url`: the repository URL used to build the image

```
config c.BinderHub.auth_enabled = Bool(False)
```

If JupyterHub authentication enabled, require user to login (don't create temporary users during launch) and start the new server for the logged in user.

```
config c.BinderHub.badge_base_url = Unicode('')
```

Base URL to use when generating launch badges

```
config c.BinderHub.banner_message = Unicode('')
```

Message to display in a banner on all pages.

The value will be inserted "as is" into a HTML `<div>` element with grey background, located at the top of the BinderHub pages. Raw HTML is supported.

```
config c.BinderHub.base_url = Unicode('/')
```

The base URL of the entire application

```
config c.BinderHub.build_cleanup_interval = Int(60)
```

Interval (in seconds) for how often stopped build pods will be deleted.

```
config c.BinderHub.build_docker_host = Unicode('/var/run/docker.sock')
```

The docker URL repo2docker should use to build the images.

Currently, only paths are supported, and they are expected to be available on all the hosts.

```
config c.BinderHub.build_image = Unicode('jupyter/repo2docker:0.10.0')
```

The repo2docker image to be used for doing builds

```
config c.BinderHub.build_max_age = Int(14400)
```

Maximum age of builds

Builds that are still running longer than this will be killed.



**config c.BinderHub.build\_memory\_limit = ByteSpecification(0)**

Max amount of memory allocated for each image build process.

0 sets no limit.

This is used as both the memory limit & request for the pod that is spawned to do the building, even though the pod itself will not be using that much memory since the docker building is happening outside the pod. However, it makes kubernetes aware of the resources being used, and lets it schedule more intelligently.

**config c.BinderHub.build\_namespace = Unicode('default')**

Kubernetes namespace to spawn build pods in.

Note that the push\_secret must refer to a secret in this namespace.

**config c.BinderHub.build\_node\_selector = Dict()**

Select the node where build pod runs on.

**config c.BinderHub.builder\_required = Bool(True)**

If binderhub should try to continue to run without a working build infrastructure.

Build infrastructure is kubernetes cluster + docker. This is useful for pure HTML/CSS/JS local development.

**config c.BinderHub.concurrent\_build\_limit = Int(32)**

The number of concurrent builds to allow.

**config c.BinderHub.config\_file = Unicode('binderhub\_config.py')**

Config file to load.

If a relative path is provided, it is taken relative to current directory

**config c.BinderHub.debug = Bool(False)**

Turn on debugging.

**config c.BinderHub.executor\_threads = Int(5)**

The number of threads to use for blocking calls

Should generally be a small number because we don't care about high concurrency here, just not blocking the webserver. This executor is not used for long-running tasks (e.g. builds).

**config c.BinderHub.extra\_footer\_scripts = Dict()**

Extra bits of JavaScript that should be loaded in footer of each page.

Only the values are set up as scripts. Keys are used only for sorting.

Omit the <script> tag. This should be primarily used for analytics code.

**config c.BinderHub.extra\_static\_path = Unicode('')**

Path to search for extra static files.

**config c.BinderHub.extra\_static\_url\_prefix = Unicode('/extra\_static/')**

Url prefix to serve extra static files.

**config c.BinderHub.google\_analytics\_code = Unicode(None)**

The Google Analytics code to use on the main page.

Note that we'll respect Do Not Track settings, despite the fact that GA does not. We will not load the GA scripts on browsers with DNT enabled.

**config c.BinderHub.google\_analytics\_domain = Unicode('auto')**

The Google Analytics domain to use on the main page.

By default this is set to 'auto', which sets it up for current domain and all subdomains. This can be set to a more restrictive domain here for better privacy

**config c.BinderHub.hub\_api\_token = Unicode('')**

API token for talking to the JupyterHub API

**config c.BinderHub.hub\_url = Unicode('')**

The base URL of the JupyterHub instance where users will run.

e.g. <https://hub.mybinder.org/>

**config c.BinderHub.image\_prefix = Unicode('')**

Prefix for all built docker images.

**If you are pushing to gcr.io, this would start with:** gcr.io/<your-project-name>/

Set according to whatever registry you are pushing to.

Defaults to "", which is probably not what you want :)

**config c.BinderHub.log\_datefmt = Unicode('%Y-%m-%d %H:%M:%S')**

The date format used by logging formatters for %(asctime)s

**config c.BinderHub.log\_format = Unicode('[%(name)s]%(highlevel)s %(message)s')**

The Logging format template

**config c.BinderHub.log\_level = Enum(30)**

Set the log level by value or name.

**config c.BinderHub.log\_tail\_lines = Int(100)**

Limit number of log lines to show when connecting to an already running build.

**config c.BinderHub.normalized\_origin = Unicode('')**

Origin to use when emitting events. Defaults to hostname of request when empty

**config c.BinderHub.per\_repo\_quota = Int(0)**

Maximum number of concurrent users running from a given repo.

Limits the amount of Binder that can be consumed by a single repo.

0 (default) means no quotas.

**config c.BinderHub.per\_repo\_quota\_higher = Int(0)**

Maximum number of concurrent users running from a higher-quota repo.

Limits the amount of Binder that can be consumed by a single repo. This quota is a second limit for repos with special status. See the `high_quota_specs` parameter of `RepoProvider` classes for usage.

0 (default) means no quotas.

**config c.BinderHub.pod\_quota = Int(None)**

The number of concurrent pods this hub has been designed to support.

This quota is used as an indication for how much above or below the design capacity a hub is running. It is not used to reject new launch requests when usage is above the quota.

The default corresponds to no quota, 0 means the hub can't accept pods (maybe because it is in maintenance mode), and any positive integer sets the quota.

**config c.BinderHub.port = Int(8585)**

Port for the builder to listen on.

**config c.BinderHub.push\_secret = Unicode('binder-push-secret')**

A kubernetes secret object that provides credentials for pushing built images.

**config c.BinderHub.repo\_providers = Dict()**

List of Repo Providers to register and try

**config c.BinderHub.sticky\_builds = Bool(False)**

Attempt to assign builds for the same repository to the same node.

In order to speed up re-builds of a repository all its builds will be assigned to the same node in the cluster.

Note: This feature only works if you also enable docker-in-docker support.

**config c.BinderHub.template\_path = Unicode('')**

Path to search for custom jinja templates, before using the default templates.

**config c.BinderHub.template\_variables = Dict()**

Extra variables to supply to jinja templates when rendering.

**config c.BinderHub.tornado\_settings = Dict()**

additional settings to pass through to tornado.

can include things like additional headers, etc.

**config c.BinderHub.use\_named\_servers = Bool(False)**

Use named servers when authentication is enabled.

**config c.BinderHub.use\_registry = Bool(True)**

Set to true to push images to a registry & check for images in registry.

Set to false to use only local docker images. Useful when running in a single node.

**config c.BinderHub.about\_message = Unicode('')**

Additional message to display on the about page.

Will be directly inserted into the about page's source so you can use raw HTML.

**static add\_url\_prefix** (*prefix, handlers*)

add a url prefix to handlers

**config c.BinderHub.appendix = Unicode('')**

Appendix to pass to repo2docker

A multi-line string of Docker directives to run. Since the build context cannot be affected, ADD will typically not be useful.

This should be a Python string template. It will be formatted with at least the following names available:

- `binder_url`: the shareable URL for the current image (e.g. for sharing links to the current Binder)
- `repo_url`: the repository URL used to build the image

**config c.BinderHub.auth\_enabled = Bool(False)**

If JupyterHub authentication enabled, require user to login (don't create temporary users during launch) and start the new server for the logged in user.

**config c.BinderHub.badge\_base\_url = Unicode('')**

Base URL to use when generating launch badges

**config c.BinderHub.banner\_message = Unicode('')**

Message to display in a banner on all pages.

The value will be inserted "as is" into a HTML <div> element with grey background, located at the top of the BinderHub pages. Raw HTML is supported.

**config c.BinderHub.base\_url = Unicode('/')**

The base URL of the entire application

**config c.BinderHub.build\_cleanup\_interval = Int(60)**

Interval (in seconds) for how often stopped build pods will be deleted.

**config c.BinderHub.build\_docker\_host = Unicode('/var/run/docker.sock')**  
The docker URL repo2docker should use to build the images.

Currently, only paths are supported, and they are expected to be available on all the hosts.

**config c.BinderHub.build\_image = Unicode('jupyter/repo2docker:0.10.0')**  
The repo2docker image to be used for doing builds

**config c.BinderHub.build\_max\_age = Int(14400)**  
Maximum age of builds

Builds that are still running longer than this will be killed.

**config c.BinderHub.build\_memory\_limit = ByteSpecification(0)**  
Max amount of memory allocated for each image build process.

0 sets no limit.

This is used as both the memory limit & request for the pod that is spawned to do the building, even though the pod itself will not be using that much memory since the docker building is happening outside the pod. However, it makes kubernetes aware of the resources being used, and lets it schedule more intelligently.

**config c.BinderHub.build\_namespace = Unicode('default')**  
Kubernetes namespace to spawn build pods in.

Note that the push\_secret must refer to a secret in this namespace.

**config c.BinderHub.build\_node\_selector = Dict()**  
Select the node where build pod runs on.

**config c.BinderHub.builder\_required = Bool(True)**  
If binderhub should try to continue to run without a working build infrastructure.

Build infrastructure is kubernetes cluster + docker. This is useful for pure HTML/CSS/JS local development.

**config c.BinderHub.concurrent\_build\_limit = Int(32)**  
The number of concurrent builds to allow.

**config c.BinderHub.config\_file = Unicode('binderhub\_config.py')**  
Config file to load.

If a relative path is provided, it is taken relative to current directory

**config c.BinderHub.debug = Bool(False)**  
Turn on debugging.

**config c.BinderHub.executor\_threads = Int(5)**  
The number of threads to use for blocking calls

Should generally be a small number because we don't care about high concurrency here, just not blocking the webserver. This executor is not used for long-running tasks (e.g. builds).

**config c.BinderHub.extra\_footer\_scripts = Dict()**  
Extra bits of JavaScript that should be loaded in footer of each page.

Only the values are set up as scripts. Keys are used only for sorting.

Omit the <script> tag. This should be primarily used for analytics code.

**config c.BinderHub.extra\_static\_path = Unicode('')**  
Path to search for extra static files.

**config c.BinderHub.extra\_static\_url\_prefix = Unicode('/extra\_static/')**  
Url prefix to serve extra static files.

**config c.BinderHub.google\_analytics\_code = Unicode(None)**

The Google Analytics code to use on the main page.

Note that we'll respect Do Not Track settings, despite the fact that GA does not. We will not load the GA scripts on browsers with DNT enabled.

**config c.BinderHub.google\_analytics\_domain = Unicode('auto')**

The Google Analytics domain to use on the main page.

By default this is set to 'auto', which sets it up for current domain and all subdomains. This can be set to a more restrictive domain here for better privacy

**config c.BinderHub.hub\_api\_token = Unicode('')**

API token for talking to the JupyterHub API

**config c.BinderHub.hub\_url = Unicode('')**

The base URL of the JupyterHub instance where users will run.

e.g. <https://hub.mybinder.org/>

**config c.BinderHub.image\_prefix = Unicode('')**

Prefix for all built docker images.

**If you are pushing to gcr.io, this would start with:** `gcr.io/<your-project-name>/`

Set according to whatever registry you are pushing to.

Defaults to "", which is probably not what you want :)

**initialize (\*args, \*\*kwargs)**

Load configuration settings.

**config c.BinderHub.log\_tail\_lines = Int(100)**

Limit number of log lines to show when connecting to an already running build.

**config c.BinderHub.normalized\_origin = Unicode('')**

Origin to use when emitting events. Defaults to hostname of request when empty

**config c.BinderHub.per\_repo\_quota = Int(0)**

Maximum number of concurrent users running from a given repo.

Limits the amount of Binder that can be consumed by a single repo.

0 (default) means no quotas.

**config c.BinderHub.per\_repo\_quota\_higher = Int(0)**

Maximum number of concurrent users running from a higher-quota repo.

Limits the amount of Binder that can be consumed by a single repo. This quota is a second limit for repos with special status. See the `high_quota_specs` parameter of `RepoProvider` classes for usage.

0 (default) means no quotas.

**config c.BinderHub.pod\_quota = Int(None)**

The number of concurrent pods this hub has been designed to support.

This quota is used as an indication for how much above or below the design capacity a hub is running. It is not used to reject new launch requests when usage is above the quota.

The default corresponds to no quota, 0 means the hub can't accept pods (maybe because it is in maintenance mode), and any positive integer sets the quota.

**config c.BinderHub.port = Int(8585)**

Port for the builder to listen on.

**config c.BinderHub.push\_secret = Unicode('binder-push-secret')**

A kubernetes secret object that provides credentials for pushing built images.

**config c.BinderHub.repo\_providers = Dict()**

List of Repo Providers to register and try

**start (run\_loop=True)**

Start the app mainloop.

Override in subclasses.

**config c.BinderHub.sticky\_builds = Bool(False)**

Attempt to assign builds for the same repository to the same node.

In order to speed up re-builds of a repository all its builds will be assigned to the same node in the cluster.

Note: This feature only works if you also enable docker-in-docker support.

**config c.BinderHub.template\_path = Unicode('')**

Path to search for custom jinja templates, before using the default templates.

**config c.BinderHub.template\_variables = Dict()**

Extra variables to supply to jinja templates when rendering.

**config c.BinderHub.tornado\_settings = Dict()**

additional settings to pass through to tornado.

can include things like additional headers, etc.

**config c.BinderHub.use\_named\_servers = Bool(False)**

Use named servers when authentication is enabled.

**config c.BinderHub.use\_registry = Bool(True)**

Set to true to push images to a registry & check for images in registry.

Set to false to use only local docker images. Useful when running in a single node.

**watch\_build\_pods()**

Watch build pods

Every build\_cleanup\_interval: - delete stopped build pods - delete running build pods older than build\_max\_age

## **build**

### **Module: binderhub.build**

Contains build of a docker image from a git repository.

### **Build**

```
class binderhub.build.Build(q, api, name, namespace, repo_url, ref, git_credentials,
                             build_image, image_name, push_secret, memory_limit,
                             docker_host, node_selector, appendix="", log_tail_lines=100,
                             sticky_builds=False)
```

Represents a build of a git repository into a docker image.

This ultimately maps to a single pod on a kubernetes cluster. Many different build objects can point to this single pod and perform operations on the pod. The code in this class needs to be careful and take this into account.

For example, operations a Build object tries might not succeed because another Build object pointing to the same pod might have done something else. This should be handled gracefully, and the build object should reflect the state of the pod as quickly as possible.

**name** The `name` should be unique and immutable since it is used to sync to the pod. The `name` should be unique for a `(repo_url, ref)` tuple, and the same tuple should correspond to the same `name`. This allows use of the locking provided by k8s API instead of having to invent our own locking code.

**cleanup()**

Delete a kubernetes pod.

**classmethod cleanup\_builds** (*kube, namespace, max\_age*)

Delete stopped build pods and build pods that have aged out

**get\_affinity()**

Determine the affinity term for the build pod.

There are a two affinity strategies, which one is used depends on how the BinderHub is configured.

In the default setup the affinity of each build pod is an “anti-affinity” which causes the pods to prefer to schedule on separate nodes.

In a setup with docker-in-docker enabled pods for a particular repository prefer to schedule on the same node in order to reuse the docker layer cache of previous builds.

**get\_cmd()**

Get the cmd to run to build the image

**progress** (*kind, obj*)

Put the current action item into the queue for execution.

**stop()**

Stop watching a build

**stream\_logs()**

Stream a pod’s logs

**submit()**

Submit a build pod to create the image for the repository.

## builder

**Module:** `binderhub.builder`

Handlers for working with version control services (i.e. GitHub) for builds.

### BuildHandler

**class** `binderhub.builder.BuildHandler` (*application: tornado.web.Application, request: tornado.httputil.HTTPServerRequest, \*\*kwargs*)

A handler for working with GitHub.

**emit** (*data*)

Emit an eventstream event

**get** (*provider\_prefix, \_unescaped\_spec*)

Get a built image for a given spec and repo provider.

Different repo providers will require different spec information. This function relies on the functionality of the tornado GET request.

### Parameters

- **provider\_prefix** (*str*) – the nickname for a repo provider (i.e. ‘gh’)
- **spec** – specifies information needed by the repo provider (i.e. user, repo, ref, etc.)

#### **keep\_alive** ()

Constantly emit keepalive events

So that intermediate proxies don’t terminate an idle connection

#### **launch** (*kube, provider*)

Ask JupyterHub to launch the image.

#### **on\_finish** ()

Stop keepalive when finish has been called

#### **send\_error** (*status\_code, \*\*kwargs*)

event stream cannot set an error code, so send an error event

## main

### Module: `binderhub.main`

Main handler classes for requests

#### **MainHandler**

```
class binderhub.main.MainHandler (application: tornado.web.Application, request: tornado.httputil.HTTPServerRequest, **kwargs)
```

Main handler for requests

#### **ParameterizedMainHandler**

```
class binderhub.main.ParameterizedMainHandler (application: tornado.web.Application, request: tornado.httputil.HTTPServerRequest, **kwargs)
```

Main handler that allows different parameter settings

#### **LegacyRedirectHandler**

```
class binderhub.main.LegacyRedirectHandler (application: tornado.web.Application, request: tornado.httputil.HTTPServerRequest, **kwargs)
```

Redirect handler from legacy Binder

## registry

`binderhub.repoproviders`



**Module: binderhub.registry**

Interaction with the Docker Registry

**DockerRegistry**

```
class binderhub.registry.DockerRegistry (**kwargs)
```

**repoproviders****Module: binderhub.repoproviders**

Classes for Repo providers

Subclass the base class, RepoProvider, to support different version control services and providers.

**Note:** When adding a new repo provider, add it to the allowed values for repo providers in event-schemas/launch.json.

**RepoProvider**

```
class binderhub.repoproviders.RepoProvider (**kwargs)
```

Base class for a repo provider

```
config c.RepoProvider.banned_specs = List()
```

List of specs to blacklist building.

Should be a list of regexes (not regex objects) that match specs which should be blacklisted

```
config c.RepoProvider.git_credentials = Unicode('')
```

Credentials (if any) to pass to git when cloning.

```
config c.RepoProvider.high_quota_specs = List()
```

List of specs to assign a higher quota limit.

Should be a list of regexes (not regex objects) that match specs which should have a higher quota

```
config c.RepoProvider.spec_config = List()
```

List of dictionaries that define per-repository configuration.

Each item in the list is a dictionary with two keys:

**pattern** [string] defines a regex pattern (not a regex object) that matches specs.

**config** [dict] a dictionary of “config\_name: config\_value” pairs that will be applied to any repository that matches **pattern**

```
config c.RepoProvider.banned_specs = List()
```

List of specs to blacklist building.

Should be a list of regexes (not regex objects) that match specs which should be blacklisted

```
get_build_slug()
```

Return a unique build slug

```
get_repo_url()
```

Return the git clone-able repo URL

**get\_resolved\_ref\_url()**  
Return the URL of repository at this commit in history

**get\_resolved\_spec()**  
Return the spec with resolved ref.

**config c.RepoProvider.git\_credentials = Unicode('')**  
Credentials (if any) to pass to git when cloning.

**has\_higher\_quota()**  
Return true if the given spec has a higher quota

**config c.RepoProvider.high\_quota\_specs = List()**  
List of specs to assign a higher quota limit.  
  
Should be a list of regexes (not regex objects) that match specs which should have a higher quota

**is\_banned()**  
Return true if the given spec has been banned

**repo\_config(settings)**  
Return configuration for this repository.

**config c.RepoProvider.spec\_config = List()**  
List of dictionaries that define per-repository configuration.  
  
Each item in the list is a dictionary with two keys:

- pattern** [string] defines a regex pattern (not a regex object) that matches specs.
- config** [dict] a dictionary of “config\_name: config\_value” pairs that will be applied to any repository that matches **pattern**

### GitHubRepoProvider

**class binderhub.repoproviders.GitHubRepoProvider(\*args, \*\*kwargs)**  
Repo provider for the GitHub service

**config c.GitHubRepoProvider.access\_token = Unicode('')**  
GitHub access token for authentication with the GitHub API  
  
Loaded from GITHUB\_ACCESS\_TOKEN env by default.

**config c.GitHubRepoProvider.api\_base\_path = Unicode('https://api.{hostname}')**  
The base path of the GitHub API  
  
Only necessary if not github.com, e.g. GitHub Enterprise.  
  
Can use {hostname} for substitution, e.g. ‘https://{hostname}/api/v3’

**config c.GitHubRepoProvider.banned\_specs = List()**  
List of specs to blacklist building.  
  
Should be a list of regexes (not regex objects) that match specs which should be blacklisted

**config c.GitHubRepoProvider.client\_id = Unicode('')**  
GitHub client id for authentication with the GitHub API  
  
For use with client\_secret. Loaded from GITHUB\_CLIENT\_ID env by default.

**config c.GitHubRepoProvider.client\_secret = Unicode('')**  
GitHub client secret for authentication with the GitHub API  
  
For use with client\_id. Loaded from GITHUB\_CLIENT\_SECRET env by default.

**config c.GitHubRepoProvider.git\_credentials = Unicode('')**  
 Credentials (if any) to pass to git when cloning.

**config c.GitHubRepoProvider.high\_quota\_specs = List()**  
 List of specs to assign a higher quota limit.

Should be a list of regexes (not regex objects) that match specs which should have a higher quota

**config c.GitHubRepoProvider.hostname = Unicode('github.com')**  
 The GitHub hostname to use

Only necessary if not github.com, e.g. GitHub Enterprise.

**config c.GitHubRepoProvider.spec\_config = List()**  
 List of dictionaries that define per-repository configuration.

Each item in the list is a dictionary with two keys:

- pattern** [string] defines a regex pattern (not a regex object) that matches specs.
- config** [dict] a dictionary of “config\_name: config\_value” pairs that will be applied to any repository that matches **pattern**

**config c.GitHubRepoProvider.access\_token = Unicode('')**  
 GitHub access token for authentication with the GitHub API

Loaded from GITHUB\_ACCESS\_TOKEN env by default.

**config c.GitHubRepoProvider.api\_base\_path = Unicode('https://api.{hostname}')**  
 The base path of the GitHub API

Only necessary if not github.com, e.g. GitHub Enterprise.

Can use {hostname} for substitution, e.g. ‘https://{hostname}/api/v3’

**config c.GitHubRepoProvider.client\_id = Unicode('')**  
 GitHub client id for authentication with the GitHub API

For use with client\_secret. Loaded from GITHUB\_CLIENT\_ID env by default.

**config c.GitHubRepoProvider.client\_secret = Unicode('')**  
 GitHub client secret for authentication with the GitHub API

For use with client\_id. Loaded from GITHUB\_CLIENT\_SECRET env by default.

**get\_build\_slug()**  
 Return a unique build slug

**get\_repo\_url()**  
 Return the git clone-able repo URL

**get\_resolved\_ref\_url()**  
 Return the URL of repository at this commit in history

**get\_resolved\_spec()**  
 Return the spec with resolved ref.

**config c.GitHubRepoProvider.hostname = Unicode('github.com')**  
 The GitHub hostname to use

Only necessary if not github.com, e.g. GitHub Enterprise.



---

## The BinderHub community

---

The BinderHub community includes members of organizations deploying their own BinderHubs, as well as members of the broader Jupyter and Binder communities.

This section contains a collection of resources for and about the BinderHub community.

### 6.1 Community

The BinderHub community includes members of organizations deploying their own BinderHubs, as well as members of the broader Jupyter and Binder communities.

This section contains a collection of resources for and about the BinderHub community.

#### 6.1.1 The BinderHub Federation

While it may seem like `mybinder.org` is a single website, it is in fact a **federation** of teams that deploy public BinderHubs to serve the community. This page lists the BinderHubs that currently help power `mybinder.org`.

Visiting `mybinder.org` will randomly redirect you to one of the following BinderHubs.

---

**Note:** If your organization is interested in becoming part of the BinderHub federation, check out *[Joining the BinderHub Federation](#)*.

---

#### Members of the BinderHub Federation

Here is a list of the current members of the BinderHub federation:

## Joining the BinderHub Federation

Behind `mybinder.org` is a **federation of BinderHubs**. This means that there are several independent hubs that each serve a fraction of the traffic created by people clicking links pointing to `mybinder.org`. Anyone (a company, university or individual) is welcome to deploy a BinderHub that forms part of the federation.

Adding a new BinderHub to the federation requires a mix of two kinds of resources: compute and human power to operate the hub. The two extremes of this mixture are:

- You donate compute power that the `mybinder.org` team **has full control over**, which means you don't have to be involved in day to day operations
- You donate compute power over which the `mybinder.org` team **does not** have full control which means you are also responsible for day to day operations of the BinderHub.

## Things to consider when deciding to join the Binder federation

If you're interested in joining the federation of BinderHubs, consider the following questions:

1. **How much time will this take?** Answering this question depends largely on how comfortable you are deploying and maintaining your own BinderHub. If you are fairly comfortable, it won't take much time. Otherwise, it may be a good idea to gain some experience in running a BinderHub first - perhaps by helping with the `mybinder.org` deployment!
2. **Is there any kind of service agreement?** Not really. We expect that any member of the BinderHub federation will be committed to keeping their BinderHub running with a reasonable uptime, but we don't have any legal framework to enforce this. Use your best judgment when deciding if you'd like to join the BinderHub federation - if you can confidently say your BinderHub will be up the large majority of the time, then that's fine.
3. **What kind of cloud resources would I need?** This depends on how many you have :) We can increase or decrease the percentage of `mybinder.org` traffic that goes to your BinderHub based on what you can handle.
4. **I'm still interested, what should I do next to join?** If you'd still like to join the BinderHub federation, see [How to join the BinderHub Federation](#).

## How to join the BinderHub Federation

If you've read through [Things to consider when deciding to join the Binder federation](#) and would like to join the BinderHub federation, please reach out to the Binder team by opening an issue at [the mybinder.org repository](#). Mention that you'd like to join the federation, what kind of computational resources you have, and what kind of human resources you have for maintaining the BinderHub deployment.

The next step is for you to tell us where your BinderHub lives. We'll assign a sub-domain of `mybinder.org` (e.g. `ovh.mybinder.org`) that points to your BinderHub. Finally, we'll change the routing configuration so that some percentage of traffic to `mybinder.org` is directed to your BinderHub! The last step is to tell everybody how awesome you are, and to add your deployment to [Members of the BinderHub Federation](#) page.

## The BinderHub Federation FAQ

### Can I deploy a BinderHub *both* for the federation and for my own community?

Yes! BinderHub can be deployed either as a public service (such as at `mybinder.org`), or for a more restricted community. Serving a smaller community means you can expose users to more resources or allow access to privileged data.

If you'd like to both serve a more specific population of users *and* support the public mybinder.org federation, we recommend running two BinderHubs in parallel with one another. You can do this on the same Kubernetes cluster if you wish, and you'd configure each BinderHub according to the resources and access that you want to provide.

### Who is currently in the BinderHub federation?

The current list of BinderHubs that are contributing to mybinder.org can be found at [Members of the BinderHub Federation](#).

### Does the BinderHub federation share Docker images?

Currently, the federation does *not* share Docker images for repositories. This means that you might have to build your repository a few times (one for each BinderHub that serves your images). We know that this adds some extra waiting for many folks, and if you have any suggestions for how we can speed up this process please [open an issue](#) in the BinderHub repository!

## 6.1.2 BinderHub Deployments

BinderHub is open-source technology that can be deployed anywhere that Kubernetes is deployed. The Binder community hopes that it will be used for many applications in research and education. As new organizations adopt BinderHub, we'll update this page in order to provide inspiration to others who wish to do so.

If you or your organization has set up a BinderHub that isn't listed here, please [open an issue](#) on our GitHub repository to discuss adding it!

### GESIS - Leibniz-Institute for the Social Sciences

Deployed on bare-metal using `kubeadm`.

- [Deployment repository](#)
- [BinderHub / JupyterHub links](#)

### Pangeo - A community platform for big data geoscience

Pangeo-Binder allows users to perform computations using distributed computing resources via the [dask-kubernetes](#) package. Read more about the [Pangeo project here](#). Pangeo-Binder is deployed on Google Cloud Platform using Google Kubernetes Engine (GKE).

dask-kubernetes: <https://dask-kubernetes.readthedocs.io/en/latest/> Pangeo project here: <https://pangeo.io/>

- [Deployment repository](#)
- [BinderHub / JupyterHub links](#)
- [Pangeo-Binder documentation](#)

### OVH - A public BinderHub

Deployed on [OVH](#)'s Kubernetes platform. This hub was the first hub to join the Binder Federation, a global network of general purpose, public BinderHubs.

- [Deployment repository](#)

- [BinderHub link](#)



### **b**

- `binderhub.app`, 36
- `binderhub.build`, 42
- `binderhub.builder`, 43
- `binderhub.main`, 44
- `binderhub.registry`, 45
- `binderhub.repoproviders`, 45



## A

`add_url_prefix()` (*binderhub.app.BinderHub static method*), 39

## B

`BinderHub` (class in *binderhub.app*), 36  
*binderhub.app* (module), 36  
*binderhub.build* (module), 42  
*binderhub.builder* (module), 43  
*binderhub.main* (module), 44  
*binderhub.registry* (module), 45  
*binderhub.repoproviders* (module), 45  
`Build` (class in *binderhub.build*), 42  
`BuildHandler` (class in *binderhub.builder*), 43

## C

`cleanup()` (*binderhub.build.Build method*), 43  
`cleanup_builds()` (*binderhub.build.Build class method*), 43

## D

`DockerRegistry` (class in *binderhub.registry*), 45

## E

`emit()` (*binderhub.builder.BuildHandler method*), 43

## G

`get()` (*binderhub.builder.BuildHandler method*), 43  
`get_affinity()` (*binderhub.build.Build method*), 43  
`get_build_slug()` (*binderhub.repoproviders.GitHubRepoProvider method*), 47  
`get_build_slug()` (*binderhub.repoproviders.RepoProvider method*), 45  
`get_cmd()` (*binderhub.build.Build method*), 43  
`get_repo_url()` (*binderhub.repoproviders.GitHubRepoProvider method*), 47

`get_repo_url()` (*binderhub.repoproviders.RepoProvider method*), 45

`get_resolved_ref_url()` (*binderhub.repoproviders.GitHubRepoProvider method*), 47

`get_resolved_ref_url()` (*binderhub.repoproviders.RepoProvider method*), 45

`get_resolved_spec()` (*binderhub.repoproviders.GitHubRepoProvider method*), 47

`get_resolved_spec()` (*binderhub.repoproviders.RepoProvider method*), 46

`GitHubRepoProvider` (class in *binderhub.repoproviders*), 46

## H

`has_higher_quota()` (*binderhub.repoproviders.RepoProvider method*), 46

## I

`initialize()` (*binderhub.app.BinderHub method*), 41

`is_banned()` (*binderhub.repoproviders.RepoProvider method*), 46

## K

`keep_alive()` (*binderhub.builder.BuildHandler method*), 44

## L

`launch()` (*binderhub.builder.BuildHandler method*), 44

`LegacyRedirectHandler` (class in *binderhub.main*), 44

## M

`MainHandler` (*class in binderhub.main*), [44](#)

## O

`on_finish()` (*binderhub.builder.BuildHandler method*), [44](#)

## P

`ParameterizedMainHandler` (*class in binderhub.main*), [44](#)

`progress()` (*binderhub.build.Build method*), [43](#)

## R

`repo_config()` (*binderhub.repoproviders.RepoProvider method*), [46](#)

`RepoProvider` (*class in binderhub.repoproviders*), [45](#)

## S

`send_error()` (*binderhub.builder.BuildHandler method*), [44](#)

`start()` (*binderhub.app.BinderHub method*), [42](#)

`stop()` (*binderhub.build.Build method*), [43](#)

`stream_logs()` (*binderhub.build.Build method*), [43](#)

`submit()` (*binderhub.build.Build method*), [43](#)

## W

`watch_build_pods()` (*binderhub.app.BinderHub method*), [42](#)