Notebooks Documentation

EGI Foundation

Sep 22, 2021

Contents:

1 Getting support **3**

1.1 Notebooks Service . . . 3
1.2 Data Management . . . 4
1.3 Integration with other services . . . 4
1.4 Early adopter communities and Customisations . . . 4
1.5 FAQ (Frequently Asked Questions) . . . 5
1.6 Training instance . . . 5
1.7 Internal Service Architecture . . . 10
1.8 Service Operations . . . 15

The more you go in data analysis, the more you understand that the most suitable tool for coding and visualizing is not a pure code, or SQL IDE, or even simplified data manipulation diagrams (aka workflows or jobs). From some point you realize that you need a mix of these all – that's what "notebook" platforms are, with Jupyter being the most popular notebook software out there.

Notebooks is an 'as a Service' environment based on the Jupyter technology, offering a browser-based, scalable tool for interactive data analysis. The Notebooks environment provides users with notebooks where they can combine text, mathematics, computations and rich media output. EGI Notebooks is a multi-user service and can scale to multiple servers based on the EGI Cloud service.

CHAPTER 1

Getting support

You can use the EGI Helpdesk to contact us for support or any additional service requests.

1.1 Notebooks Service

1.1.1 Unique Features

EGI Notebooks provides the well-known Jupyter interface for notebooks with the following added features:

- Integration with EGI Check-in for authentication, login with any EduGAIN or social accounts (e.g. Google, Facebook)
- Persistent storage associated to each user, available in the notebooks environment.
- Customisable with new notebook environments, expose any existing notebook to your users.
- Runs on EGI e-Infrastructure so can easily use EGI compute and storage from your notebooks.

1.1.2 Service Modes

We offer different service modes depending on your needs:

- Individual users can use the centrally operated service from EGI. Users, after lightweight approval, can login, write and play and re-play notebooks. Notebooks can use storage and compute capacity from the access.egi.eu Virtual Organisation. Request access via EGI marketplace.
- User communities can have their customised EGI Notebooks service instance. EGI offers consultancy and support, as well as can operate the setup. Contact support@egi.eu to make an arrangement. A community specific setup allows the community to use the community's own Virtual Organisation (i.e. federated compute and storage sites) for Jupyter, add custom libraries into Jupyter (e.g. discipline-specific analysis libraries) or have fine grained control on who can access the instance (based on the information available to the EGI Check-in AAI service).

1.2 Data Management

1GB persistent storage for the notebooks is available as your home directory. **Please note that files stored on any other folder will be lost when your notebook server is closed (which can happen if there is no activity for more than 1 hour!**).

If you need to increase your persistent storage space, open a GGUS ticket to the Notebooks Support Unit

Access to other kinds of persistent storage for community specific instances that can be tailored to your specific needs and available storage systems.

1.2.1 Getting your data in

Your notebooks have full outgoing internet connectivity so you can connect to any external service to bring data in for analysis. We are evaluating integration with EOSC-hub services for facilitating the access to input data, with EGI DataHub as first target. Please contact support@egi.eu if you are interested in other I/O integrations.

1.2.2 Deposit output data

As with input data, you can connect to any external service to deposit the notebooks output.

1.2.3 Interfacing with EUDAT B2DROP

The Notebooks service is interoperable with the EUDAT B2DROP service, allowing a user to access files stored under his/her B2DROP account from the Notebooks. To use this feature you should sign up for a B2DROP account, upload files into it, then register the account in your Notebooks session. User guide about the steps is coming soon.

1.3 Integration with other services

Notebooks running on EGI can access other existing computing and storage services. The centrally operated EGI Notebooks instance is using resources from the access.egi.eu Virtual Organisation. We are open to suggestions on which services you would like to access to create guidelines and extend the service with tools to ease these tasks.

1.4 Early adopter communities and Customisations

1.4.1 AGINFRA+/D4Science

An instance of the EGI notebooks is deployed for the AGINFRA+ project and made available via selected D4Science VREs. Besides the features described above, this instance has been further customised to support:

- Embedding the EGI notebooks interface into the community we portal, no separate web browser windows or tab to access the notebooks functionality.
- Integration of AAI between Notebooks and the community portal for single-sign on. Enabled users are automatically recognised by the notebooks.
- Access to other VRE services from the notebooks using their personal token easily available in the notebooks environment.
- File sharing between notebooks and the community web portal file space.

1.4.2 Other communities

- OpenDreamKit Open Digital Research Environment Toolkit for the Advancement of Mathematics
- IFREMER: Marine sciences

1.5 FAQ (Frequently Asked Questions)

1.5.1 How do I install library *X*?

You can install new software easily on the notebooks using `conda` or `pip`. The `%conda` and `%pip` magics can be used in a cell of your notebooks to do so, e.g. installing `rdkit`:

```
%conda install rdkit
```

Once installed you can import the libray as usual.

Note: Any modifications to the libraries/software of your notebooks will be lost when your notebook server is stoped (automatically after 1 hour of inactivity)!

1.5.2 Can I request library *X* to be installed permanently?

Yes! Just let us know what are your needs. You can contact us via:

- Opening a ticket in the EGI Helpdesk, or
- Creating a Github Issue

We will analyse your request and get back to you.

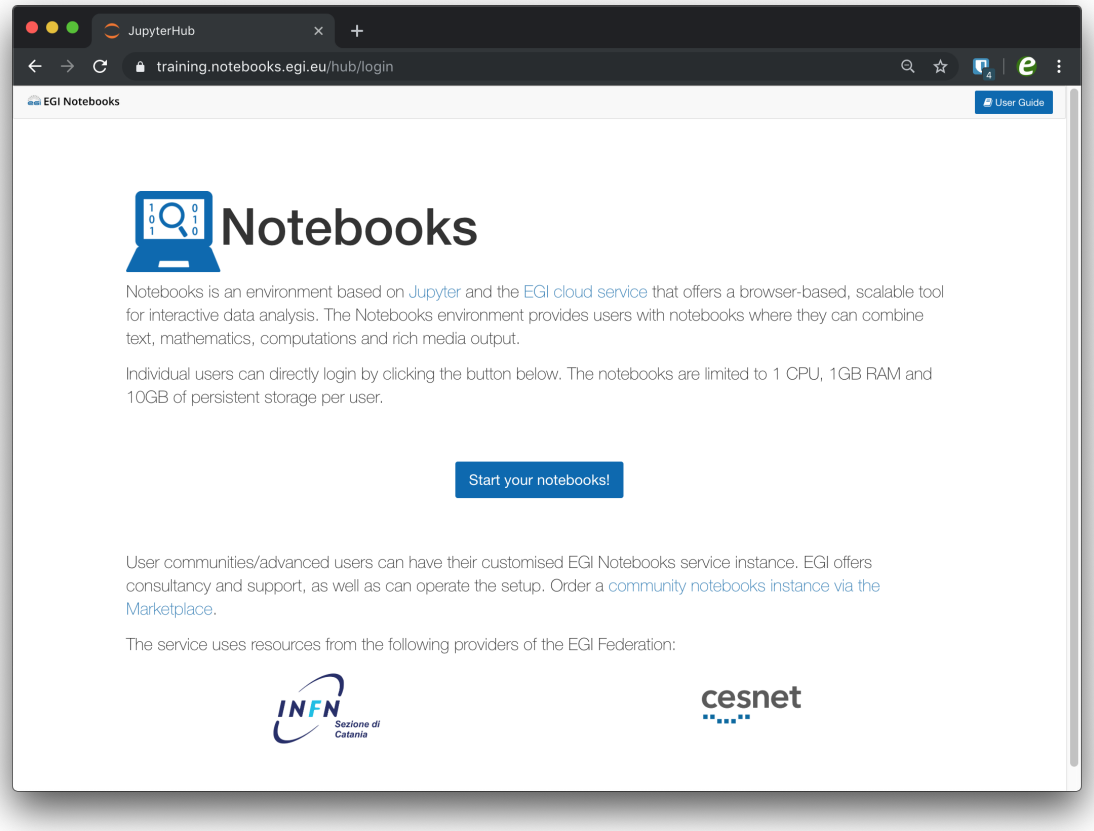
1.6 Training instance

EGI provides a training instance of the Notebooks service for training events. This instance may not use the same software version as in production and may not be always available.

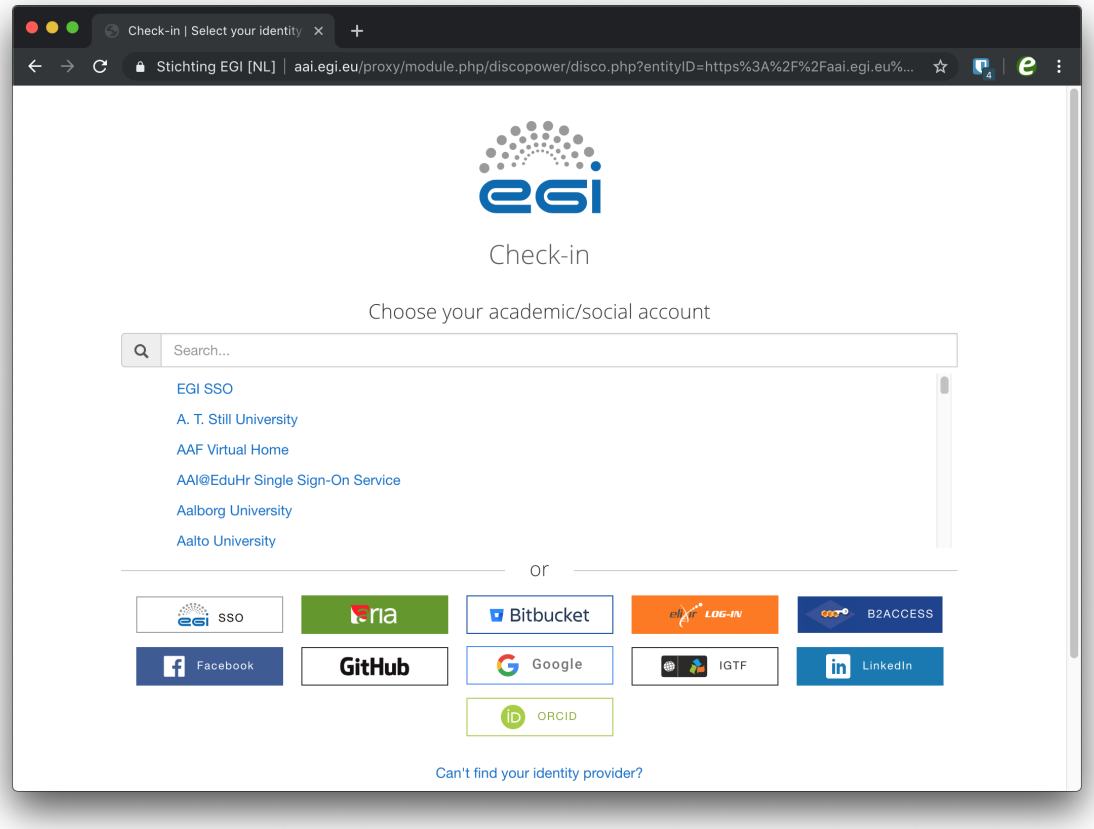
Note: The training environment for the Notebooks is available at https://training.notebooks.egi.eu.

To get started:

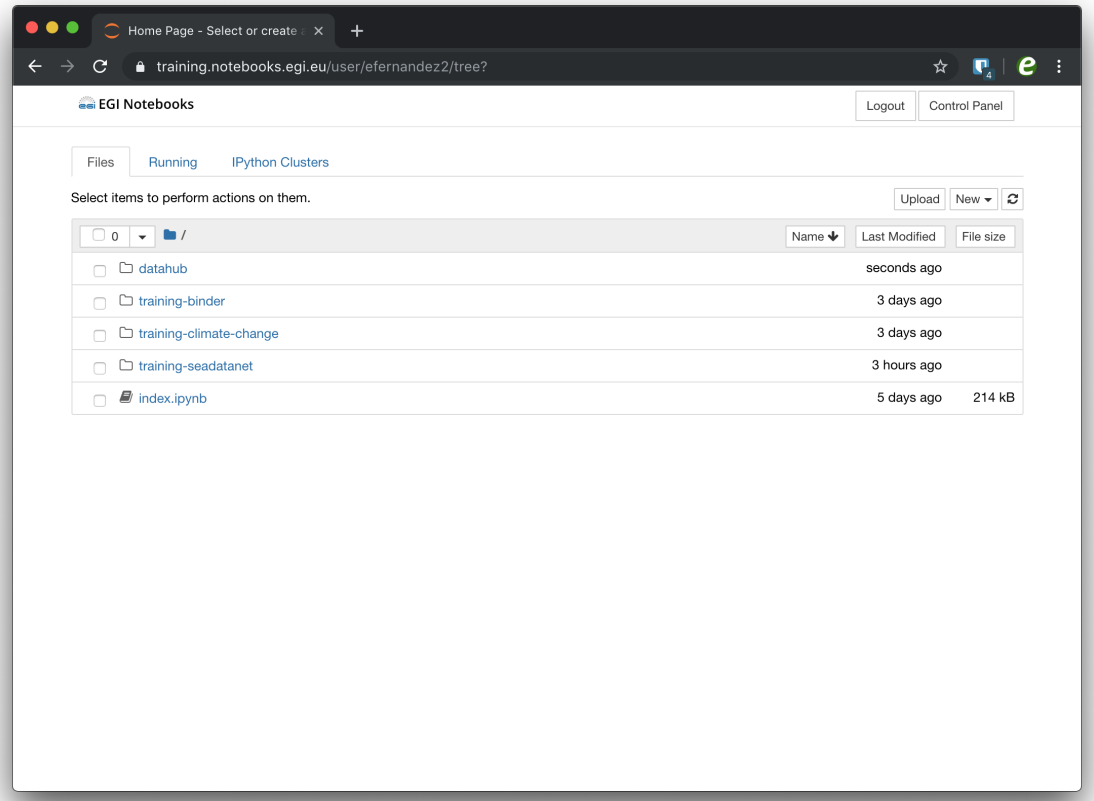
1. Go to https://training.notebooks.egi.eu
2. Start the authentication process by clicking on **Start your notebooks!** button



3. Select the Identity Provider you belong to from the discovery page. If this is the first time you access an EGI service, Check-in will guide you through a registration process.

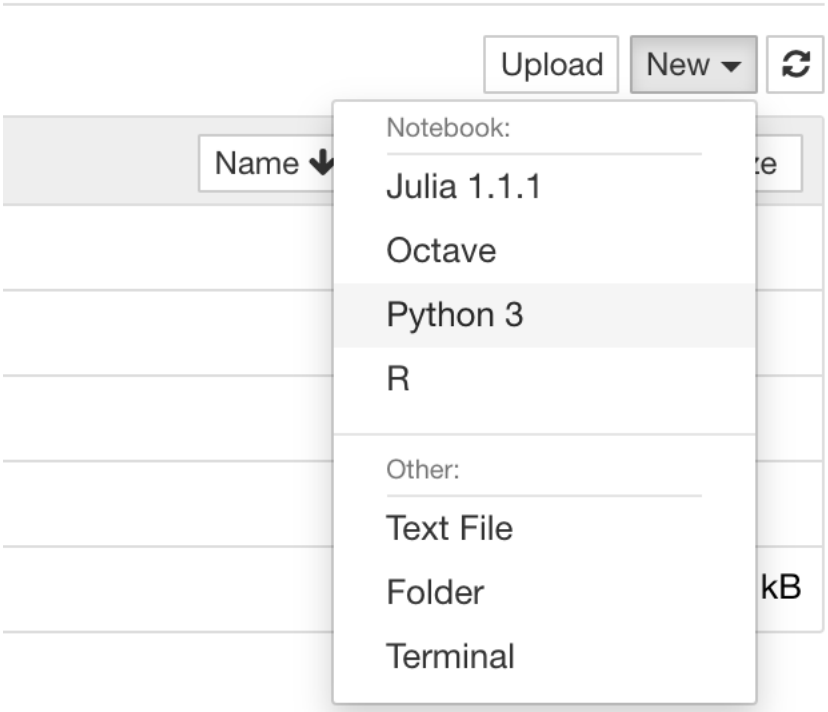


4. You will see the Jupyter interface once your personal server is started



1.6.1 Launching a notebook

Click on the New > Python 3 option to launch your notebook with Python 3 kernel. When you create this notebook, a new tab will be presented with a notebook named *Untitled.ipynb*. You can easily rename it by right-clicking on the current name.



Structure of a notebook

The notebook consists of a sequence of cells. A cell is a multiline text input field, and its contents can be executed by using `Shift-Enter`, or by clicking either the *"Play"* button in the toolbar, or Cell -> Run in the menu bar.

The execution behaviour of a cell is determined by the cell's type.

There are three types of cells: cells, markdown, and raw cells. Every cell starts off being a code cell, but its type can be changed by using a drop-down on the toolbar (which will be "Code", initially).

Code cells

A code cell allows you to edit and write new code, with full syntax highlighting and tab completion. The programming language you use depends on the kernel.

When a code cell is executed, its content is sent to the kernel associated with the notebook. The results that are returned from this computation are then displayed in the notebook as the cell's output. The output is not limited to text, with many other possible forms of output are also possible, including figures and HTML tables.

Markdown cells

You can document the computational process in a literate way, alternating descriptive text with code, using rich text. This is accomplished by marking up text with the Markdown language. The corresponding cells are called Markdown

cells. The Markdown language provides a simple way to perform this text markup, that is, to specify which parts of the text should be emphasized (italics), bold, form lists, etc.

If you want to provide structure for your document, you can also use markdown headings. Markdown headings consist of 1 to 6 hash # signs followed by a space and the title of your section. The markdown heading will be converted to a clickable link for a section of the notebook. It is also used as a hint when exporting to other document formats, like PDF.

When a Markdown cell is executed, the Markdown code is converted into the corresponding formatted rich text. Markdown allows arbitrary HTML code for formatting.

Raw cells

Raw cells provide a place in which you can write output directly. Raw cells are not evaluated by the notebook.

Keyboard shortcuts

All actions in the notebook can be performed with the mouse, but keyboard shortcuts are also available for the most common ones. These are some of the most common:

- `Shift-Enter`: run cell. Execute the current cell, show any output, and jump to the next cell below. If `Shift-Enter` is invoked on the last cell, it creates a new cell below. This is equivalent to clicking the Cell -> Run menu item, or the Play button in the toolbar.
- `Esc`: Command mode. In command mode, you can navigate around the notebook using keyboard shortcuts.
- `Enter` : Edit mode. In edit mode, you can edit text in cells.

1.6.2 Hands-on

We pre-populate your home directory with some sample notebooks to get started, below you can find links to other notebooks that we have used in past trainings that may be useful to explore the system:

1. A very basic notebook to get started
2. Getting data and doing a simple plot.
3. Connect to NOAA's GrADS Data Server to plot wind speed.
4. Installing new libraries.
5. Interact with Check-in

1.7 Internal Service Architecture

The EGI Notebooks service relies on the following technologies to provide its functionality:

- JupyterHub with custom EGI Check-in oauthentication configured to spawn pods on Kubernetes.
- Kubernetes as container orchestration platform running on top of EGI Cloud resources. Within the service it is in charge of managing the allocated resources and providing the right abstraction to deploy the containers that build the service. Resources are provided by EGI Federated Cloud providers, including persistent storage for users notebooks.
- CA authority to allocate recognised certificates for the HTTPS server
- Prometheus for monitoring resource consumption.

- Specific EGI hooks for monitoring, accounting and backup.
- VO-Specific storage/Big data facilities or any pluggable tools into the notebooks environment can be added to community specific instances.

1.7.1 Kubernetes

A Kubernetes (k8s) cluster deployed into a resource provider is in charge of managing the containers that will provide the service. On this cluster there are:

- 1 master node that manages the whole cluster
- Support for load balancer or alternatively 1 or more edge nodes with a public IP and corresponding public DNS name (e.g. notebooks.egi.eu) where a k8s ingress HTTP reverse proxy redirects requests from user to other components of the service. The HTTP server has a valid certificate from one CA recognised at most browsers (e.g. Let's Encrypt).
- 1 or more nodes that host the JupyterHub server, the notebooks servers where the users will run their notebooks. Hub is deployed using the JupyterHub helm charts. These nodes should have enough capacity to run as many concurrent user notebooks as needed. Main constraint is usually memory.
- Support for Kubernetes PersistentVolumeClaims for storing the persistent folders. Default EGI-Notebooks installation uses NFS, but any other volume type with ReadWriteOnce capabilities can be used.
- Prometheus installation to monitor the usage of resources so accounting records are generated.

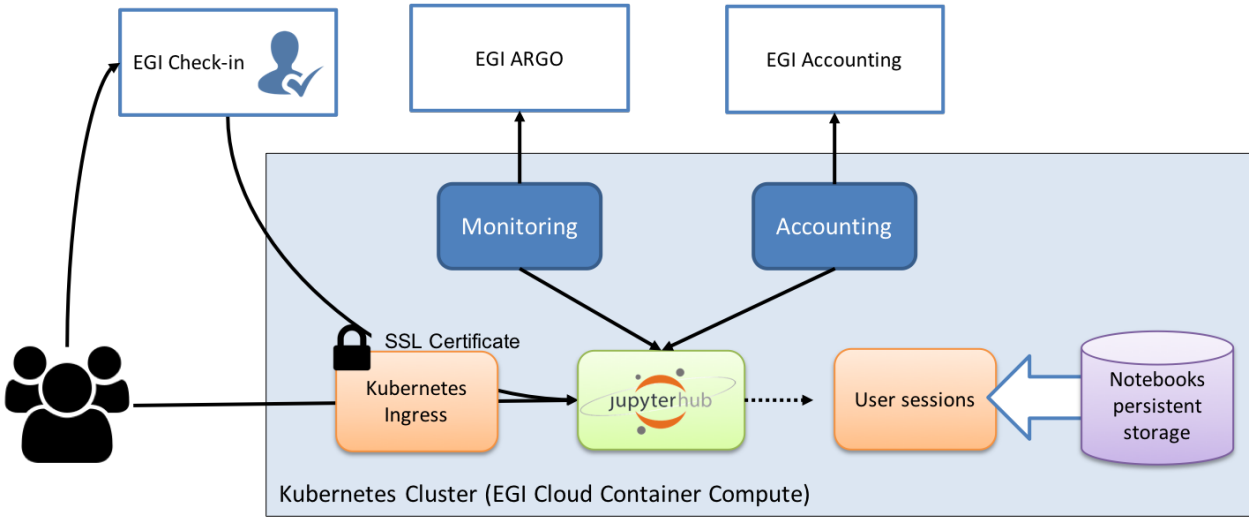
All communication with the user goes via HTTPS and the service only needs a publicly accessible entry point (public IP with resolvable name)

Monitoring and accounting are provided by hooking into the respective monitoring and accounting EGI services.

There are no specific hardware requirements and the whole environment can run on commodity virtual machines.

1.7.2 EGI Customisations

EGI Notebooks is deployed as a set of customisations of the JupyterHub helm charts.



Authentication

EGI Check-in can be easily configured as a OAuth2.0 provider for JupyterHub's oauthenticator. See below a sample configuration for the helm chart using Check-in production environment:

```
hub:
  extraEnv:
    OAUTH2_AUTHORIZE_URL: https://aai.egi.eu/oidc/authorize
    OAUTH2_TOKEN_URL: https://aai.egi.eu/oidc/token
    OAUTH_CALLBACK_URL: https://<your host>/hub/oauth_callback

auth:
  type: custom
  custom:
    className: oauthenticator.generic.GenericOAuthenticator
    config:
      login_service: "EGI Check-in"
      client_id: "<your client id>"
      client_secret: "<your client secret>"
      oauth_callback_url: "https://<your host>/hub/oauth_callback"
      username_key: "sub"
      token_url: "https://aai.egi.eu/oidc/token"
      userdata_url: "https://aai.egi.eu/oidc/userinfo"
      scope: ["openid", "profile", "email", "eduperson_scoped_affiliation",
↪"eduperson_entitlement"]
```

To simplify the configuration and to add refresh capabilities to the credentials, we have created a new EGI Check-in authenticator that can be configued as follows:

```
auth:
  state:
    enabled: true
    cryptoKey: <some unique crypto key>
  type: custom
  custom:
    className: oauthenticator.egicheckin.EGICheckinAuthenticator
    config:
      client_id: "<your client id>"
      client_secret: "<your client secret>"
      oauth_callback_url: "https://<your host>/hub/oauth_callback"
      scope: ["openid", "profile", "email", "offline_access", "eduperson_scoped_
↪affiliation", "eduperson_entitlement"]
```

The `auth.state` configuration allows to store refresh tokens for the users that will allow to get up-to-date valid credentials as needed.

Accounting

> **Warning:** documentation is not yet final!

Accounting module generates VM-like accounting records for each of the notebooks started at the service. It's available as a *helm chart <https://egi-foundation.github.io/egi-notebooks-chart/>_* that can be deployed in the same namespace as the JupyterHub chart. The only needed configuration for the chart is an IGTF-recognised certificate for the host registered in GOCDB as accounting.

```
ssm:
  hostcert: |-
    <hostcert>
  hostkey: |-
    <hostkey>
```

Monitoring

Monitoring is performed by trying to execute a user notebook every hour. This is accomplished by registering a new service in the hub that has admin permissions. Monitoring is then deployed as a *helm chart <https://egi-foundation.github.io/egi-notebooks-chart/>_* that must be deployed in the same namespace as the JupyterHub chart. Configuration of JupyterHub must include this section:

```
hub:
  services:
    status:
       url: "http://status-web/"
       admin: true
       apiToken: "<a unique API token>"
```

Likewise the monitoring chart is configured as follows:

```
service:
  api_token: "<same API token as above>"
```

Docker images

Our service relies on custom images for the hub and the single-user notebooks. Dockerfiles are available at EGI Notebooks images git repository and automatically build for every commit pushed to the repo to eginotebooks @ dockerhub.

Hub image

Builds from the JupyterHub k8s-hub image and adds:

- EGI and D4Science authenticators
- EGISpawner
- EGI look and feel for the login page

Single-user image

Builds from Jupyter datasicence-notebook and adds a wide range of libraries as requested by users of the services. We are currently looking into alternatives for better managing this image with CVMFS as a possible solution.

Sample helm configuration

If you want to build your own EGI Notebooks instance, you can start from the following sample configuration and adapt to your needs by setting:

- secret tokens (for `proxy.secretToken`, `hub.services.status.api_token`, `auth.state.cryptoKey`). They can be generated with `openssl rand -hex 32`.
- A valid host name (`<your notebooks host>` below) that resolves to your Kubernetes Ingress
- Valid EGI Check-in client credentials, these can be obtained by creating a new client at EGI AAI OpenID Connect Provider. When moving to EGI Check-in production environment, make sure to remove the `hub.extraEnv.EGICHECKIN_HOST` variable.

```
---
proxy:
  secretToken: "<some secret>"
  service:
    type: NodePort

ingress:
  enabled: true
  annotations:
    kubernetes.io/tls-acme: "true"
  hosts: [<your notebooks host>]
  tls:
  - hosts:
    - <your notebooks host>
    secretName: acme-tls-notebooks
    enabled: true
    hosts: [<your notebooks host>]

singleuser:
  storage:
    capacity: 1Gi
    dynamic:
      pvcNameTemplate: claim-{userid}{servername}
      volumeNameTemplate: vol-{userid}{servername}
      storageAccessModes: ["ReadWriteMany"]
  memory:
    limit: 1G
    guarantee: 512M
  cpu:
    limit: 2
    guarantee: .02
  defaultUrl: "/lab"
  image:
    name: eginotebooks/single-user
    tag: c1b2a2a

hub:
  image:
    name: eginotebooks/hub
    tag: c1b2a2a
  extraConfig:
    enable-lab: |-
      c.KubeSpawner.cmd = ['jupyter-labhub']
    volume-handling: |-
      from egispawner.spawner import EGISpawner
      c.JupyterHub.spawner_class = EGISpawner
  extraEnv:
    JUPYTER_ENABLE_LAB: 1
    EGICHECKIN_HOST: aai-dev.egi.eu
```

(continues on next page)

(continued from previous page)

```
  services:
    status:
       url: "http://status-web/"
       admin: true
       api_token: "<monitor token>"

auth:
  type: custom
  state:
    enabled: true
    cryptoKey: "<a unique crypto key>"
  admin:
    access: true
    users: [<list of EGI Check-in users with admin powers>]
  custom:
    className: oauthenticator.egicheckin.EGICheckinAuthenticator
    config:
      client_id: "<your egi checkin_client_id>"
      client_secret: "<your egi checkin_client_secret>"
      oauth_callback_url: "https://<your notebooks host>/hub/oauth_callback"
      enable_auth_state: true
      scope: ["openid", "profile", "email", "offline_access", "eduperson_scoped_
↪affiliation", "eduperson_entitlement"]
```

1.8 Service Operations

In this section you can find the common operational activities related to keep the service available to our users.

1.8.1 Initial set-up

Notebooks VO

The resources used for the Notebooks deployments are managed with the `vo.notebooks.egi.eu` VO. Operators of the service should join the VO, check the entry at the operations portal and at AppDB.

Clients installation

In order to manage the resources you will need these tools installed on your client machine:

- `egicli` for discovering sites and managing tokens,
- `terraform` to create the VMs at the providers,
- `ansible` to configure the VMs and install kubernetes at the providers,
- `terraform-inventory` to get the list of hosts to use from terraform.

Get the configuration repo

All the configuration of the notebooks is stored at a git repo available in keybase. You'll need to be part of the `opslife` team in keybase to access. Start by cloning the repo:

```
$ git clone keybase://team/opslife/egi-notebooks
```

1.8.2 Kubernetes

We use `terraform` and `ansible` to build the cluster at one of the EGI Cloud providers. If you are building the cluster for the first time, create a new directory on your local git repository from the template, add it to the repo, and get `terraform` ready:

```
$ cp -a template <new provider>
$ git add <new provider>
$ cd <new provider>/terraform
$ terraform init
```

Using the `egicli` you can get the list of projects and their ids for a given site:

```
$ egicli endpoint projects --site CESGA
id                                Name                 enabled    site
--------------------------------  -------------------  ---------  ------
3a8e9d966e644405bf19b536adf7743d  vo.access.egi.eu     True       CESGA
916506ac136741c28e4326975eef0bff  vo.emso-eric.eu      True       CESGA
b1d2ef2cc2284c57bcde21cf4ab141e3  vo.nextgeoss.eu      True       CESGA
eb7ff20e603d471cb731bdb83a95a2b5  fedcloud.egi.eu      True       CESGA
fcaf23d103c1485694e7494a59ee5f09  vo.notebooks.egi.eu  True       CESGA
```

And with the project ID, you can obtain all the environment variables needed to interact with the OpenStack APIs of the site:

```
$ eval "$(egicli endpoint env --site CESGA --project-id␣
↪fcaf23d103c1485694e7494a59ee5f09)"
```

Now you are ready to use the openstack or terraform at the site. The token obtained is valid for 1 hour, you can refresh it at any time with:

```
$ eval "$(egicli endpoint token --site CESGA --project-id␣
↪fcaf23d103c1485694e7494a59ee5f09)"
```

First get the network IDs and pool to use for the site:

```
$ openstack network list
+--------------------------------------+-------------------------+--------------------
↪------------------+
| ID                                   | Name                    | Subnets           ␣
↪                  |
+--------------------------------------+-------------------------+--------------------
↪------------------+
| 1aaf20b6-47a1-47ef-972e-7b36872f678f | net-vo.notebooks.egi.eu | 6465a327-c261-4391-
↪a0f5-d503cc2d43d3 |
| 6174db12-932f-4ee3-bb3e-7a0ca070d8f2 | public00                | 6af8c4f3-8e2e-405d-
↪adea-c0b374c5bd99 |
+--------------------------------------+-------------------------+--------------------
↪------------------+
```

In this case we will use `public00` as the pool for public IPs and `1aaf20b6-47a1-47ef-972e-7b36872f678f` as the network ID. Check with the provider which is the right network to use. Use these values in the `terraform.tfvars` file:

```
ip_pool = "public00"
net_id  = "1aaf20b6-47a1-47ef-972e-7b36872f678f"
```

You may want to check the right flavors for your VMs and adapt other variables in `terraform.tfvars`. To get a list of flavors you can use:

```
$ openstack flavor list
+--------------------------------------+----------------+-------+------+-----------+--
↪-----+-----------+
| ID                                   | Name           |   RAM | Disk | Ephemeral |␣
↪VCPUs | Is Public |
+--------------------------------------+----------------+-------+------+-----------+--
↪-----+-----------+
| 26d14547-96f2-4751-a686-f89a9f7cd9cc | cor4mem8hd40   |  8192 |   40 |         0 | ␣
↪    4 | True      |
| 42eb9c81-e556-4b63-bc19-4c9fb735e344 | cor2mem2hd20   |  2048 |   20 |         0 | ␣
↪    2 | True      |
| 4787d9fc-3923-4fc9-b770-30966fc3baee | cor4mem4hd40   |  4096 |   40 |         0 | ␣
↪    4 | True      |
| 58586b06-7b9d-47af-b9d0-e16d49497d09 | cor24mem62hd60 | 63488 |   60 |         0 | ␣
↪   24 | True      |
| 635c739a-692f-4890-b8fd-d50963bff00e | cor1mem1hd10   |  1024 |   10 |         0 | ␣
↪    1 | True      |
| 6ba0080d-d71c-4aff-b6f9-b5a9484097f8 | small          |   512 |    2 |         0 | ␣
↪    1 | True      |
| 6e514065-9013-4ce1-908a-0dcc173125e4 | cor2mem4hd20   |  4096 |   20 |         0 | ␣
↪    2 | True      |
| 85f66ce6-0b66-4889-a0bf-df8dc23ee540 | cor1mem2hd10   |  2048 |   10 |         0 | ␣
↪    1 | True      |
| c4aa496b-4684-4a86-bd7f-3a67c04b1fa6 | cor24mem50hd50 | 51200 |   50 |         0 | ␣
↪   24 | True      |
| edac68c3-50ea-42c2-ae1d-76b8beb306b5 | test-bigHD     |  4096 |  237 |         0 | ␣
↪    2 | True      |
+--------------------------------------+----------------+-------+------+-----------+--
↪-----+-----------+
```

Finally ensure your public ssh key is also listed in the `cloud-init.yaml` file and then you are ready to deploy the cluster with:

```
$ terraform apply
```

Your VMs are up and running, it's time to get kubernetes configured and running with ansible:

```
$ cd ..   # you should be now in <new provider>
$ ANSIBLE_TRANSFORM_INVALID_GROUP_CHARS=silently TF_STATE=./terraform \
  ansible-playbook --inventory-file=$(which terraform-inventory) \
  playbooks/k8s.yaml
```

Interacting with the cluster

As the master will be on a private IP, you won't be able to directly interact with it, but you can still ssh into the VM using the ingress node as a gateway host (you can get the different hosts with `TF_STATE=./terraform terraform-inventory --inventory`)

```
$ ssh -o ProxyCommand="ssh -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/
↪null -W %h:%p -q egi@<ingress ip>" \
      -o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null egi@<master ip>
egi@k8s-master:~$ kubectl get nodes
NAME            STATUS   ROLES    AGE   VERSION
k8s-master      Ready    master   33m   v1.15.7
k8s-nfs         Ready    <none>   16m   v1.15.7
k8s-w-ingress   Ready    <none>   16m   v1.15.7
egi@k8s-master:~$ helm list
NAME                 REVISION        UPDATED                         STATUS
↪CHART                          APP VERSION     NAMESPACE
certs-man            2               Wed Jan  8 15:56:58 2020        DEPLOYED
↪cert-manager-v0.11.0           v0.11.0         cert-manager
cluster-ingress      3               Wed Jan  8 15:56:53 2020        DEPLOYED
↪nginx-ingress-1.7.0            0.24.1          kube-system
nfs-provisioner      3               Wed Jan  8 15:56:43 2020        DEPLOYED
↪nfs-client-provisioner-1.2.8   3.1.0           kube-system
```

Modifying/Destroying the cluster

You should be able to change the number of workers in the cluster and re-apply terraform to start them and then execute the playbook to get them added to the cluster.

Any changes in the master, NFS or ingress VMs should be done carfully as those will probably break the configuration of the kubernetes cluster and of any application running on top.

Destroying the cluster can be done with a single command:

```
$ terraform destroy
```

1.8.3 Notebooks deployments

Once the k8s cluster is up and running, you can deploy a notebooks instance. For each deployment you should create a file in the *deployments* directory following the template provided:

```
$ cp deployments/hub.yaml.template deployments/hub.yaml
```

Each deployment will need a domain name pointing to your ingress host, you can create one at the FedCloud dynamic DNS service.

Then you will need to create an OpenID Connect client for EGI Check-in to authorise users into the new deployment. You can create a client by going to the Check-in demo OIDC clients management. Use the following as redirect URL: `https://<your host domain name>/hub/oauth_callback`.

In the *Access* tab, add `offline_access` to the list of scopes. Save the client and take note of the client ID and client secret for later.

Finally you will also need 3 different random strings generated with `openssl rand -hex 32` that will be used as secrets in the file describing the deployment.

Go and edit the deployment description file to add this information (search for `# FIXME NEEDS INPUT` in the file to quickly get there)

For deploying the notebooks instance we will also use `ansible`:

```
$ ANSIBLE_TRANSFORM_INVALID_GROUP_CHARS=silently TF_STATE=./terraform ansible-
↪playbook \
       --inventory-file=$(which terraform-inventory) playbooks/notebooks.yaml
```

The first deployment trial may fail due to a timeout caused by the downloading of the container images needed. You can retry after a while to re-deploy.

In the master you can check the status of your deployment (the name of the deployment will be the same as the name of your local deployment file):

```
$ helm status hub
LAST DEPLOYED: Thu Jan  9 08:14:49 2020
NAMESPACE: hub
STATUS: DEPLOYED

RESOURCES:
==> v1/ServiceAccount
NAME            SECRETS  AGE
hub             1        6m46s
user-scheduler  1        3m34s

==> v1/Service
NAME          TYPE       CLUSTER-IP     EXTERNAL-IP  PORT(S)                      AGE
hub           ClusterIP  10.100.77.129  <none>       8081/TCP                     6m46s
proxy-public  NodePort   10.107.127.44  <none>       443:32083/TCP,80:30581/TCP   6m45s
proxy-api     ClusterIP  10.103.195.6   <none>       8001/TCP                     6m45s

==> v1/ConfigMap
NAME            DATA  AGE
hub-config      4     6m47s
user-scheduler  1     3m35s

==> v1/PersistentVolumeClaim
NAME        STATUS   VOLUME                CAPACITY  ACCESS MODES  STORAGECLASS  AGE
hub-db-dir  Pending  managed-nfs-storage   6m46s

==> v1/ClusterRole
NAME                              AGE
hub-user-scheduler-complementary  3m34s

==> v1/ClusterRoleBinding
NAME                              AGE
hub-user-scheduler-base           3m34s
hub-user-scheduler-complementary  3m34s

==> v1/RoleBinding
NAME  AGE
hub   6m46s

==> v1/Pod(related)
NAME                              READY  STATUS   RESTARTS  AGE
continuous-image-puller-flf5t     1/1    Running  0         3m34s
continuous-image-puller-scr49     1/1    Running  0         3m34s
hub-569596fc54-vjbms              0/1    Pending  0         3m30s
proxy-79fb6d57c5-nj8n2            1/1    Running  0         2m22s
user-scheduler-9685d654b-9zt5d    1/1    Running  0         3m30s
user-scheduler-9685d654b-k8v9p    1/1    Running  0         3m30s
```

(continues on next page)

(continued from previous page)

```
==> v1/Secret
NAME        TYPE    DATA  AGE
hub-secret  Opaque  3     6m47s

==> v1/DaemonSet
NAME                       DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE␣
↪SELECTOR  AGE
continuous-image-puller  2        2        2      2           2          <none>␣
↪  3m34s

==> v1/Deployment
NAME            DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
hub             1        1        1           0          6m45s
proxy           1        1        1           1          6m45s
user-scheduler  2        2        2           2          3m32s

==> v1/StatefulSet
NAME              DESIRED  CURRENT  AGE
user-placeholder  0        0        6m44s

==> v1beta1/Ingress
NAME        HOSTS                                ADDRESS  PORTS  AGE
jupyterhub  notebooktest.fedcloud-tf.fedcloud.eu  80, 443  6m44s

==> v1beta1/PodDisruptionBudget
NAME              MIN AVAILABLE  MAX UNAVAILABLE  ALLOWED DISRUPTIONS  AGE
hub               1              N/A              0                    6m48s
proxy             1              N/A              0                    6m48s
user-placeholder  0              N/A              0                    6m48s
user-scheduler    1              N/A              1                    6m47s

==> v1/Role
NAME  AGE
hub   6m46s


NOTES:
Thank you for installing JupyterHub!

Your release is named hub and installed into the namespace hub.

You can find if the hub and proxy is ready by doing:

kubectl --namespace=hub get pod

and watching for both those pods to be in status 'Running'.

You can find the public IP of the JupyterHub by doing:

kubectl --namespace=hub get svc proxy-public

It might take a few minutes for it to appear!

Note that this is still an alpha release! If you have questions, feel free to
1. Read the guide at https://z2jh.jupyter.org
2. Chat with us at https://gitter.im/jupyterhub/jupyterhub
```

(continues on next page)

(continued from previous page)

```
3. File issues at https://github.com/jupyterhub/zero-to-jupyterhub-k8s/issues
```

Updating a deployment

Just edit the deployment description file and run ansible again. The helm will be upgraded at the cluster.