LANA // Docs

Create advanced dashboards

LANA Process Mining allows custom visualizations (called *advanced dashboards*) to be created using either Python (called *Python dashboards*) or the R language together with Shiny (called *Shiny dashboards*).

There are three steps that need to be completed in order to create an advanced dashboard:

1. Create an entry for the advanced dashboard (in the database)
2. Upload the archived source code for the advanced dashboard
3. Connect the advanced dashboard to an event log

All the files for a given advanced dashboard must be archived into a ZIP file, without a top-level directory. The user uploading the source code of the advanced dashboard must have the *AdvancedAnalyst* role.

Advanced dashboards using R and Shiny

Shiny dashboards are implemented in the R programming language using Shiny. Those dashboards can be either full-fledged Shiny apps or interactive documents based on R Markdown that also contain Shiny widgets and outputs.

An R API for LANA Process Mining is provided by the LanaR package, which allows the Shiny dashboards to make process mining and data aggregation requests.

The book R Markdown: The Definitive Guide contains more information on turning R Markdown documents into Shiny apps. Additionally, the flexdashboard package can also be used to create interactive dashboards.

The following example shows an interactive document that uses R Markdown and the `textInput` and `renderText` Shiny widgets. This example is specific to the LANA Process Mining UI, because the currently selected log and filter, as well as the token needed for authentication are sent by the UI using the `window.postMessage()` method.

shiny_dashboard.Rmd

````
--- ①
runtime: shiny
output: html_document
---

```{r, include = FALSE} ②
require("lanar")
````
````

```
```

```{js, echo = FALSE} ③
$(document).ready(function() {
  window.addEventListener('message', function(event) {
    Shiny.onInputChange(event.data.name, event.data.value);
  });
  window.parent.postMessage('shiny-loaded', '*');
});
```

```{r, echo = FALSE, include = FALSE} ④
lanaUrl <- "cloud-backend.lanalabs.com"

textInput("traceFilterSequence", label = NULL, value = '[]')
textInput("lanaToken", label = NULL, value = '')
textInput("selectedLogId", label = NULL, value = '')
```

```{r, echo = FALSE} ⑤
renderText({
  df <- lanar::discoveredModel(traceFilterSequence = input$traceFilterSequence,
                               lanaToken = input$lanaToken,
                               logId = input$selectedLogId,
                               lanaUrl = lanaUrl)
  c("Number of cases: ", df$logStatistics$numTraces)
})
```
```

① Entrypoint, specifies that the interactive document contains Shiny components

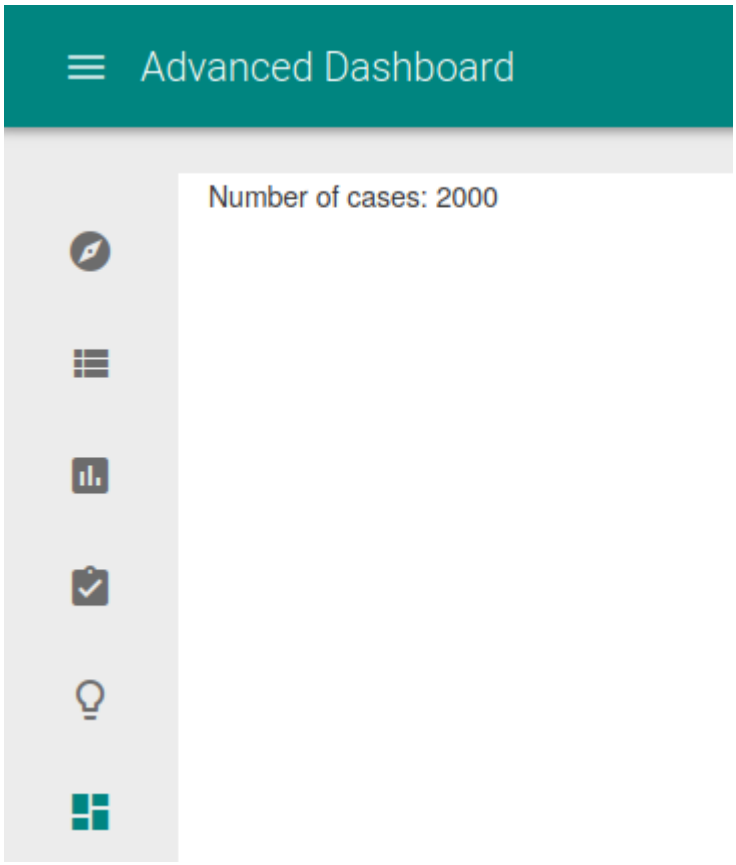
② Install the `lanar` package

③ Add an event listener for the messages sent by the LANA Process Mining UI

④ Create input fields for the log ID, filters and the token

⑤ Make a process mining request and display the number of cases from the response. More information about the response of the endpoint can be found in the Discovered model guide.

The output of the Shiny dashboard can be seen below:
```



Create a database entry

Before the Shiny dashboard source code can be uploaded, a database entry for it needs to be created.

curl

```
curl -X POST https://cloud-backend.lanalabs.com/api/v2/custom-dashboards \
  -H "Authorization: API-Key ${API_KEY}" \
  -H "Content-Type: application/json" \
  --data-raw '{
    "name": "Custom visualization using R and Shiny",
    "type": "shiny_dashboard"
  }' \
  | jq -r .id
```

Python

```
import requests


def create_custom_shiny_dashboard(api_key):
    payload = {
        'name': 'Custom visualization using Shiny',
        'type': 'shiny_dashboard'
    }

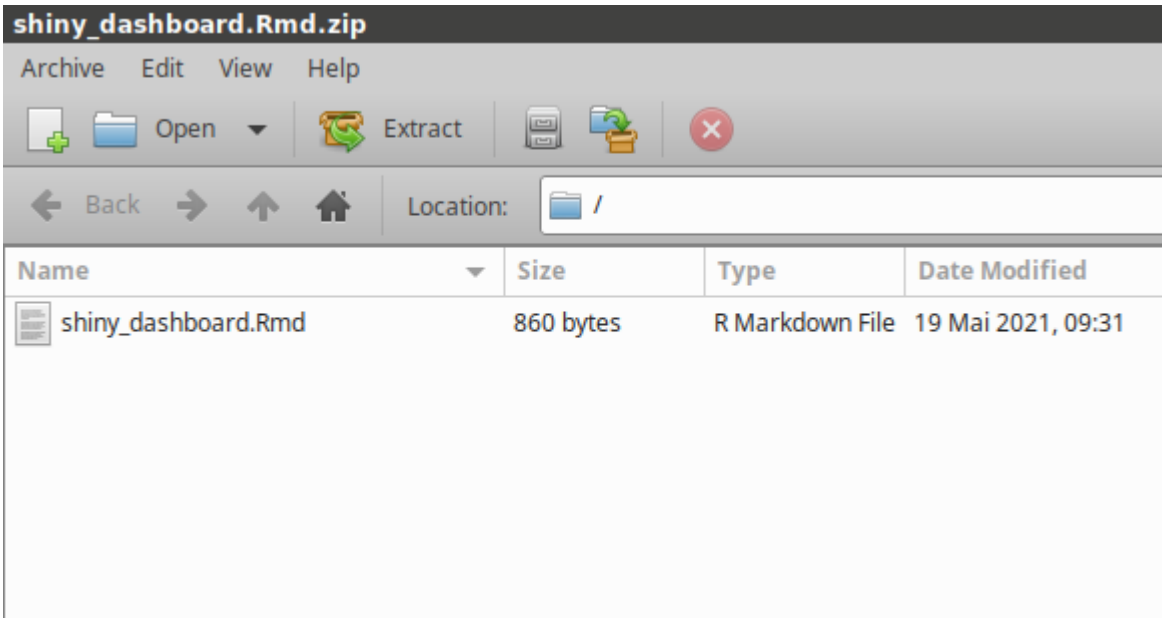
    r = requests.post(
```

```
        'https://cloud-backend.lanalabs.com/api/v2/custom-dashboards',
        headers={'API-Key': api_key},
        json=payload
    )

    return r.json()
```

Upload the source code

The Shiny dashboard can be as simple as the one shown above or it can be split into multiple files. If the Shiny dashboard is split into multiple files, the entrypoint for it must be in a file called `index.Rmd`.



curl

```
curl -X POST https://cloud-backend.lanalabs.com/api/v2/custom-
dashboards/${DASHBOARD_ID}/source \
  -H "Authorization: API-Key ${API_KEY}" \
  -F file=@"shiny_dashboard.Rmd.zip"
```

Python

```
import requests
from pathlib import Path


def upload_shiny_dashboard_app(api_key, dashboard_id, shiny_dashboard_file):
    filetype = 'application/octet-stream'
    filename = Path(shiny_dashboard_file).name

    shiny_app = [
        ('file', (filename, open(shiny_dashboard_file, 'rb'), filetype))
    ]
```

```
    r = requests.post(
        f'https://cloud-backend.lanalabs.com/api/v2/custom-
dashboards/{dashboard_id}/source',
        headers={'API-Key': api_key},
        files=shiny_app
    )

    return r
```

Connect to an event log

In order to see the advanced dashboard in the LANA Process Mining UI, the dashboard needs to be connected to an event log log. This step assumes a log has been already uploaded. The Shiny dashboard is interactive and its output will change when you add, remove or change global filters.

NOTE

How to create an event log is described in the Upload event logs guide.

curl

```
curl -X POST https://cloud-backend.lanalabs.com/api/v2/resource-connections \
  -H "Authorization: API-Key ${API_KEY}" \
  -H "Content-Type: application/json" \
  --data-raw '{
    "log_id":"'"${LOG_ID}"'",
    "custom_dashboard_id":"'"${DASHBOARD_ID}"'"
  }'
```

Python

```
import requests


def connect_dashboard_app_to_log(api_key, log_id, dashboard_id):
    payload = {
        'log_id': log_id,
        'custom_dashboard_id': dashboard_id
    }

    r = requests.post(
        'https://cloud-backend.lanalabs.com/api/v2/resource-connections',
        headers={'API-Key': api_key},
        json=payload
    )

    return r.json()
```

Advanced dashboards using Python

Python dashboards are implemented in Python. The dashboards are served as Python web applications by uWSGI.

index.py

```
def application(environ, start_response):
    body = b'<html><body>Hello, World!</body></html>' ①
    status = '200 OK' ②
    headers = [('Content-type', 'text/html')] ③
    start_response(status, headers)
    return [body]
```

① The web application must return an HTML page

② Set the response code

③ Set the response headers

Create a database entry

Before the Python dashboard source code can be uploaded, a database entry for it needs to be created.

curl

```
curl -X POST https://cloud-backend.lanalabs.com/api/v2/custom-dashboards \
  -H "Authorization: API-Key ${API_KEY}" \
  -H "Content-Type: application/json" \
  --data-raw '{
    "name": "Custom visualization using Python",
    "type": "python_dashboard"
  }' \
  | jq -r .id
```

Python

```
import requests

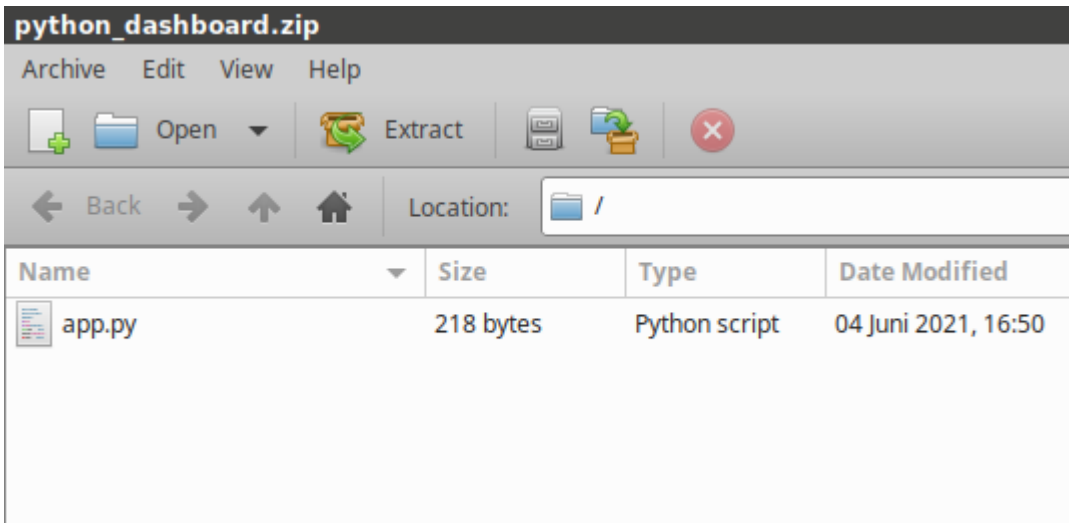

def create_custom_python_dashboard(api_key):
    payload = {
        'name': 'Custom visualization using Python',
        'type': 'python_dashboard'
    }

    r = requests.post(
        'https://cloud-backend.lanalabs.com/api/v2/custom-dashboards',
        headers={'API-Key': api_key},
        json=payload
    )

    return r.json()
```

Upload the source code

The Python dashboard can be as simple as the one shown above or it can be split into multiple files. If the Python dashboard is split into multiple file, the entrypoint for it must be in a file called `index.py`.



curl

```
curl -X POST https://cloud-backend.lanalabs.com/api/v2/custom-
dashboards/${DASHBOARD_ID}/source \
  -H "Authorization: API-Key ${API_KEY}" \
  -F file=@"python_app.zip"
```

Python

```
import requests
from pathlib import Path


def upload_python_dashboard_app(api_key, dashboard_id, python_dashboard_file):
    filetype = 'application/octet-stream'
    filename = Path(python_dashboard_file).name

    python_app = [
        ('file', (filename, open(python_dashboard_file, 'rb'), filetype))
    ]

    r = requests.post(
        f'https://cloud-backend.lanalabs.com/api/v2/custom-
dashboards/{dashboard_id}/source',
        headers={'API-Key': api_key},
        files=python_app
    )

    return r
```

Connect to an event log

In order to see the Python dashboard in the LANA Process Mining UI, the dashboard needs to be connected to an event log log. This step assumes that the event log has already been uploaded.

NOTE

How to create an event log is described in the Upload event logs guide.

curl

```
curl -X POST https://cloud-backend.lanalabs.com/api/v2/resource-connections \
  -H "Authorization: API-Key ${API_KEY}" \
  -H "Content-Type: application/json" \
  --data-raw '{
    "log_id":"'"${LOG_ID}"'",
    "custom_dashboard_id":"'"${DASHBOARD_ID}"'"
  }'
```

Python

```
import requests


def connect_dashboard_app_to_log(api_key, log_id, dashboard_id):
    payload = {
        'log_id': log_id,
        'custom_dashboard_id': dashboard_id
    }

    r = requests.post(
        'https://cloud-backend.lanalabs.com/api/v2/resource-connections',
        headers={'API-Key': api_key},
        json=payload
    )

    return r.json()
```