Please log into ETH Gitlab

If you have never logged in, your account doesn’t exist, and we can’t give you access rights to your team’s code. :(

https://gitlab.ethz.ch

VIScon Hackathon Workshop

An introduction to the VIS infrastructure

VIS infrastructure?

The VIS runs apps

VIS website

coffee statistics

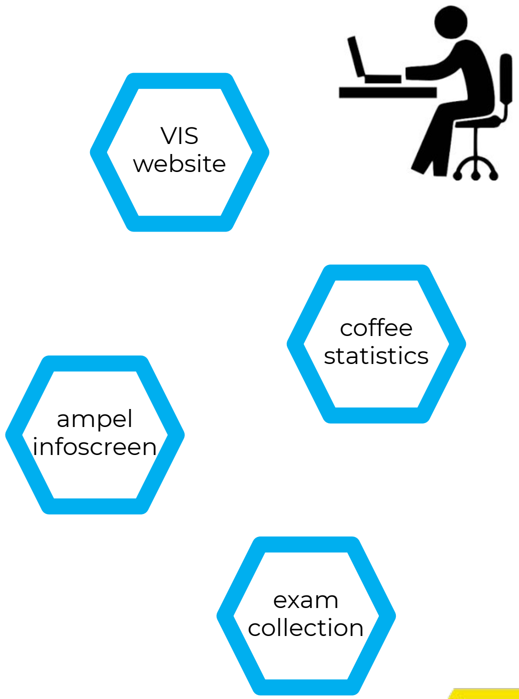
ampel infoscreen

exam collection

VIS infrastructure?

The VIS runs apps

These apps are developed by volunteers in the CAT

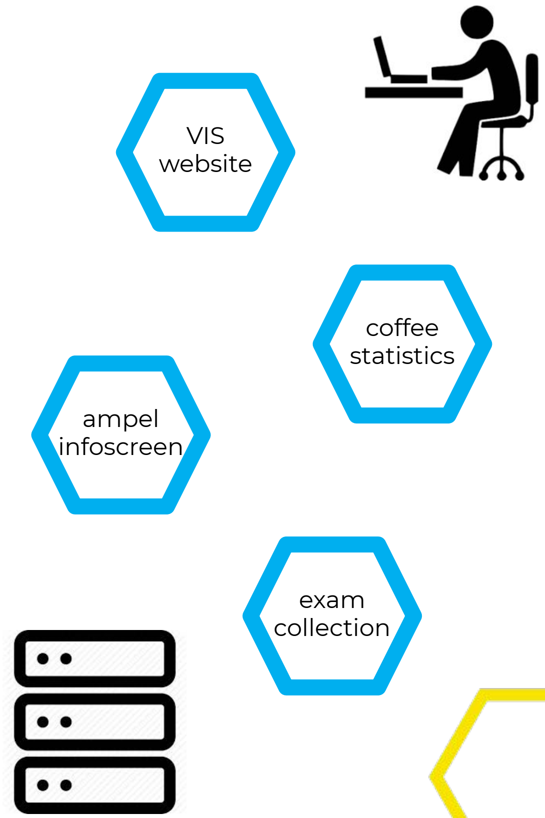


VIS infrastructure?

The VIS runs apps

These apps are developed by volunteers in the CAT

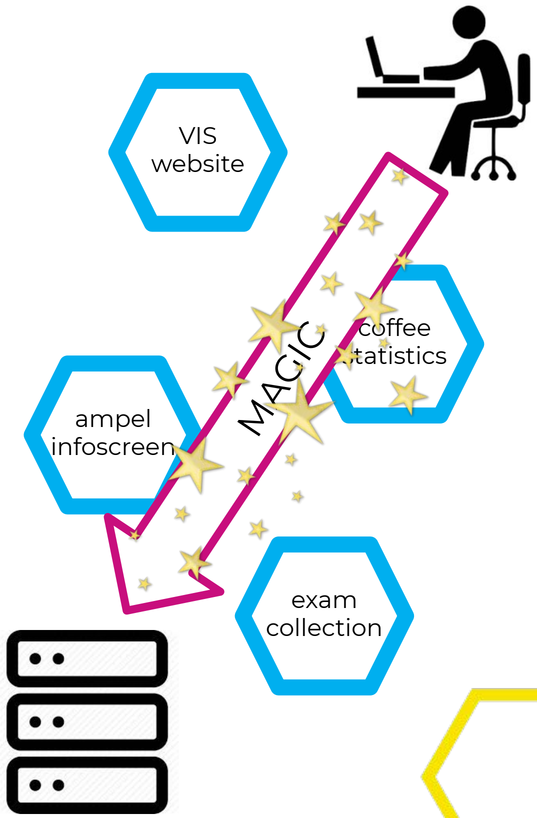
The apps run on some servers owned by VIS and maintained by the CIT



VIS infrastructure?

When the CAT members make changes, these changes are automatically compiled and uploaded to the servers (deployed).

Since this process is automatic and happens continuously, it is called **continuous deployment** (CD), or sometimes continuous integration (CI).



Why do you care?

At the VIScon Hackathon, you will build VIS apps :D

They will run on our infrastructure and make use of our continuous deployment process

This means your apps have to follow some standards - otherwise our continuous deployment will be confused.

It will be helpful for you if you know how that works ;)

The VIS App Standard ™

https://documentation.vis.ethz.ch/

- Your app runs in **docker** and it has to build on top of one of the VIS base docker images
- Your app should make use of **cinit** to start itself
- Your code's git repository contains a **CI config file**
- Your repository has the two branches "production" and "staging" - this we will prepare for you
- Your code is hosted on the ETH gitlab and connected to the VIS CI runner - this we will also prepare for you

Interactive Workshop

- Go to https://gitlab.ethz.ch/viscon19
- Find your team's repository
- Clone your repo to your local machine

The example app is a very simple python webserver that serves a webpage which just says "Hello World". We will now deploy this app onto the VIS infrastructure, just like you will do at the hackathon.

Step by step - create Dockerfile

You have coded up your app - now you want to dockerize it

For this, you need to create the Dockerfile:

```
FROM registry.vis.ethz.ch/public/base:charlie

RUN apt install -y python3

ADD src /app

EXPOSE 80
```

Step by step - create cinit file

Docker still doesn't know how to execute your app. We use cinit for that. Cinit is already present in your docker image and just needs to be configured. This is done in a yaml file, we call it cinit.yml:

```
programs:
  - name: server
    path: python3
    args:
      - "-u"
      - /app/hello_vis.py
      - 80
    user: app-user
    group: app-user
    capabilities:
      - CAP_NET_BIND_SERVICE
```

Step by step - add cinit file to docker

Don’t forget to add the cinit config file to your docker image:

```
FROM registry.vis.ethz.ch/public/base:charlie

RUN apt install -y python3

COPY cinit.yml /etc/cinit.d/demo.yml

ADD src /src
EXPOSE 80
```

Step by step - build docker image

At this point you can build your docker image

You don’t need to do this, but it can be helpful for testing whether your Dockerfile is correct.

```
docker build .
```

Step by step - create .gitlab-ci.yml file

Now we have to create the CI config file, which tells our continuous deployment process how to handle your app. The file is called .gitlab-ci.yml

You can use the template file .gitlab-ci.yaml.template - simply move it to the correct location.

Step by step - create .gitlab-ci.yml file

The top of the file looks something like this:

```
variables:
  VIS_CI_APP_NAME: "vct-x"
  VIS_CI_DEPLOYMENT_SUBDOMAIN: "x"
```

Please, **never** change your app name or subdomain. Things will break, and then Tech Support will be very sad.

Don't make Tech Support sad.

Step by step - push to gitlab

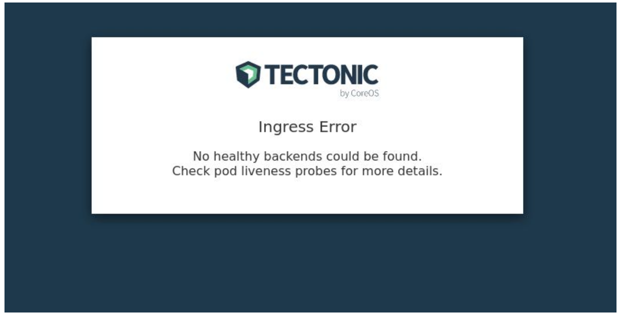
With your Dockerfile, cinit config, and .gitlab-ci.yml files ready, all you need to do is push to the repository.

Your “build pipeline” will automatically be run, and your app will be online at team-name.svis.ethz.ch

“svis.ethz.ch” is the staging environment of the vis. This is where we test our apps before moving them to “vis.ethz.ch”, where people will actually use them.

At VIScon, you will only work in staging.

If it's not working...

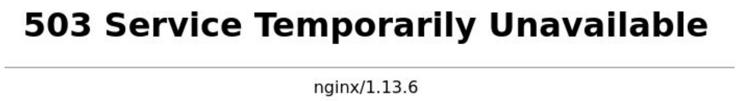

TECTONIC
by CoreOS

Ingress Error

No healthy backends could be found.
Check pod liveness probes for more details.

Your app was not deployed!

Check your .gitlab-ci.yml file and your Dockerfile. See the build logs on Gitlab.


503 Service Temporarily Unavailable

nginx/1.13.6

Your app was deployed, but it failed to start or crashed!

Check your cinit.yml file and your code. See the app logs on logs.vis.ethz.ch

Check your app’s build logs

The logs of the build process of your application can be viewed on Gitlab.

Navigate to “CI/CD” -> “Pipelines”. You can click on any of the stages (little circles) to view the logs of that stage.

There’s a lot of “machine poop” but somewhere near the top (under the green “do_default_build” line) you should find the output of the `docker build` command.

Check your app's logs

The logs of your deployed application are available at

https://logs.vis.ethz.ch

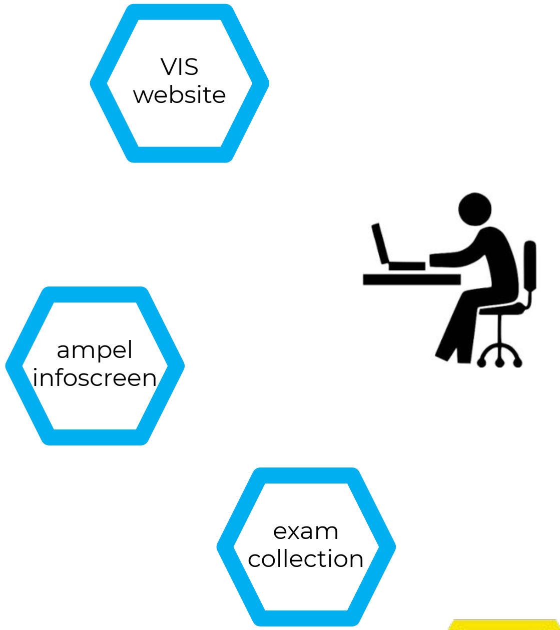
You can log in with your VIS credentials (not NETHZ!), or if you don't have any, you can use "testuser" with password "testuser" (available only during VIScon ;))

Note that it can take up to a few minutes for your logs to appear.

Questions so far?

visdev

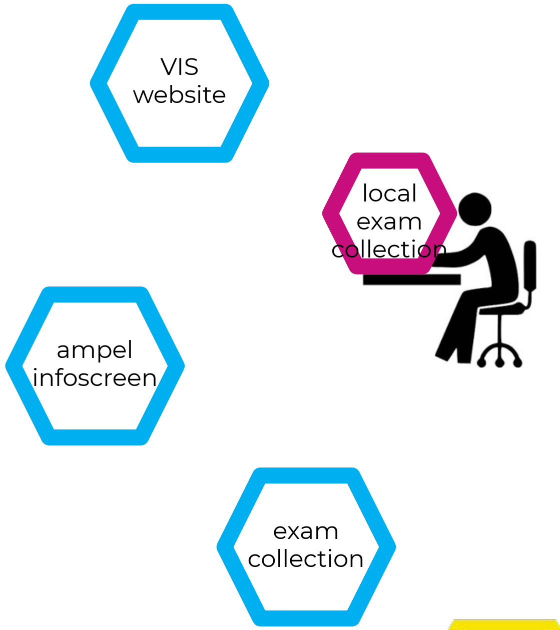
How can you try out your app locally, without pushing and waiting for the continuous deployment process?



visdev

How can you try out your app locally, without pushing and waiting for the continuous deployment process?

visdev allows you to run any app that adheres to the VIS app standard on your local machine.



Step by step - install visdev

https://documentation.vis.ethz.ch/visdev.html

To install visdev, run the following (in a separate folder):

```
git clone https://gitlab.ethz.ch/vis/cat/visdev.git
cd visdev
python3 -m venv .venv
. .venv/bin/activate
pip install --upgrade .
```

Step by step - run your app

You now have access to the visdev command. Switch back to your forked test repository. You can now run

```
visdev test .
```

This will start your app, and you can access it at

```
http://localhost:8080
```

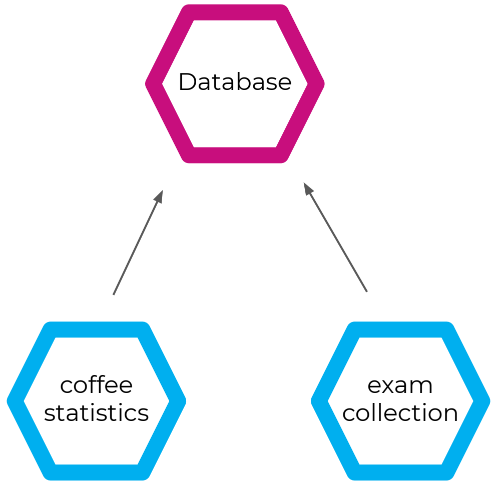
Questions so far?

Database

You cannot store files in a docker container - they will disappear on the next restart.

If you need to store data permanently, you should use a database.

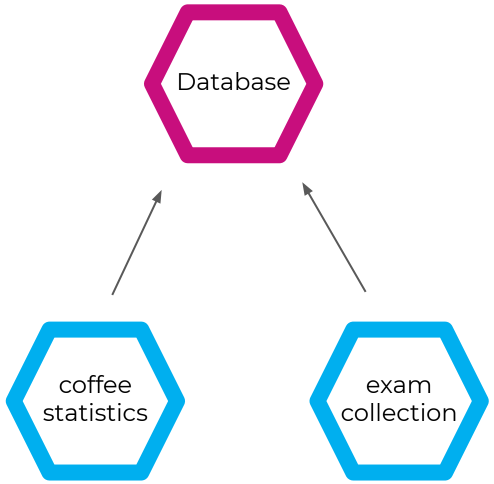
The VIS infrastructure offers a database that all apps can use.



Database

If you want to get access to a database, you need to tell the continuous deployment process about it.

The VIS offers two database flavours: PostgreSQL and MySQL (MariaDB).



Step by step - modify .gitlab-ci.yml

To tell the deployment process about your database needs, you have to modify .gitlab-ci.yml:

```
variables:
  VIS_CI_APP_NAME: "vct-x"
  VIS_CI_DEPLOYMENT_SUBDOMAIN: "x"
  VIS_CI_ENABLE_POSTGRES: "true"
```

We recommend PostgreSQL. If, however, you really want to use MySQL (MariaDB) instead, you can ask Tech Support about it during the Hackathon.

Step by step - use the database

To actually interact with your database from within your code, you can use a library for PostgreSQL. You will most certainly find something for your preferred programming language.

For this workshop, we've provided an example in hello_vis_database.py. Please rename that file to hello_vis.py (replacing the previous file).

Step by step - add new dependencies

Our Python code is now using additional libraries. We need to make sure they are available inside our Docker container.

```
FROM registry.vis.ethz.ch/public/base:charlie

RUN apt install -y python3 python3-pip

COPY requirements.txt /requirements.txt
RUN pip3 install -r /requirements.txt

COPY cinit.yml /etc/cinit.d/demo.yml
ADD src /app
EXPOSE 80
```

Step by step - environment variables

One last piece is missing: To interact with the database, you need to know on which URL and port it is, and you need an user and password to log in.

Your app receives all this information through environment variables. That way, it is always up to date.

```
RUNTIME_POSTGRES_DB_SERVER
RUNTIME_POSTGRES_DB_PORT
RUNTIME_POSTGRES_DB_NAME
RUNTIME_POSTGRES_DB_USER
RUNTIME_POSTGRES_DB_PW
```

Step by step - add env vars to cinit

The variables are injected in your container by the continuous deployment process, but cinit removes them again for safety reasons!

You need to tell cinit that you want these variables. At the end of your cinit config, add:

```
capabilities:
  - CAP_NET_BIND_SERVICE
env:
  - RUNTIME_POSTGRES_DB_SERVER:
  - RUNTIME_POSTGRES_DB_PORT:
  - RUNTIME_POSTGRES_DB_NAME:
  - RUNTIME_POSTGRES_DB_USER:
  - RUNTIME_POSTGRES_DB_PW:
```

Step by step - read the env vars

In Python, you can now read environment variables like so:

```
import os

POSTGRESQL_HOST = os.environ["RUNTIME_POSTGRES_DB_SERVER"]
POSTGRESQL_PORT = os.environ["RUNTIME_POSTGRES_DB_PORT"]
POSTGRESQL_DB = os.environ["RUNTIME_POSTGRES_DB_NAME"]
POSTGRESQL_USER = os.environ["RUNTIME_POSTGRES_DB_USER"]
POSTGRESQL_PASS = os.environ["RUNTIME_POSTGRES_DB_PW"]
```

Step by step - test it locally

Now, your app isn’t able to run unless these environment variables exist. How can you test it locally?

Don’t worry! visdev got you covered!

If you use visdev, everything will be set up right, and you will even have access to a local database for testing.

Try running the app locally with visdev test and writing some values to the database. They will persist even if you restart the application!

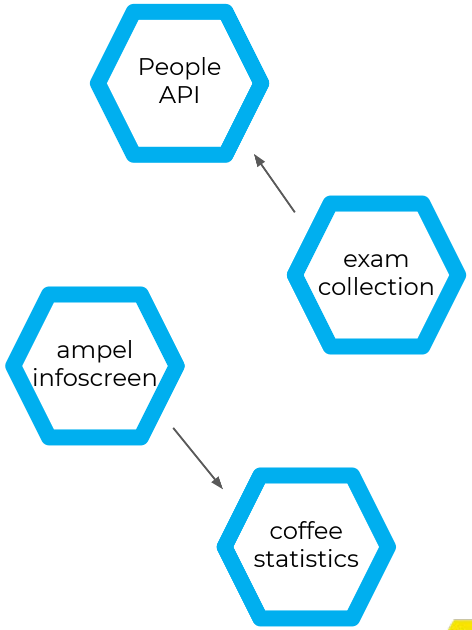
Questions so far?

Servis

Some of the VIS apps provide useful information to other VIS apps.

These apps are said to offer a “servis”, and they all use the same API conventions.

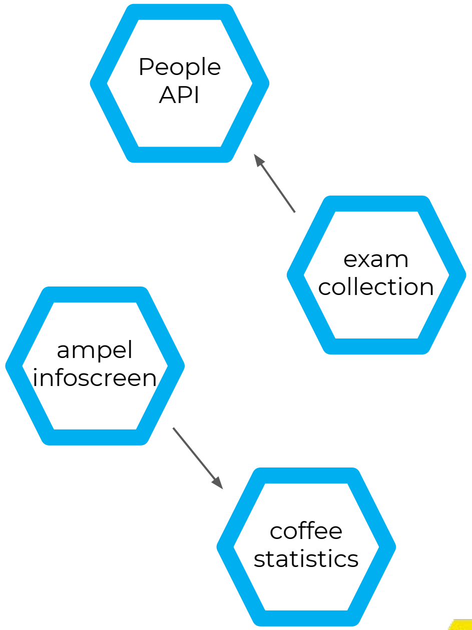
Apps communicate with servises using the GRPC protocol.



Servis

If you want to use an existing servis (e.g. the people API, which you can use to create a nethz login for your app), you need to take some extra steps.

The continuous deployment process needs to know about all servis-es you use.



https://documentation.vis.ethz.ch/servis.html

Step by step - add servis to .gitlab-ci.yml

Let’s add the servis information to .gitlab-ci.yml:

```
variables:
  VIS_CI_APP_NAME: "vct-x"
  VIS_CI_DEPLOYMENT_SUBDOMAIN: "x"
  VIS_CI_SERVIS_DEPENDENCIES: "people-api"
  VIS_CI_SERVIS_PYTHON_OUT: "./src/"
```

We use “PYTHON_OUT” because our project is in Python.

We also need to add the following line:

```
script:                 # this is already present
  - do_servis_generate
  - do_default_build    # this is already present
```

Step by step

Now, when your app is deployed, it receives access to the people-api servis.

But how do you *use* the people-api?

Step by step - generate servis files

To work with the servis in your code, you will need to generate some helper files. This is done using the servis CLI application, which you should now download:

https://ser.vis.ethz.ch/cli

Put the executable in your home directory. Then, run inside your repository:

```
~/servis add people-api
~/servis generate
```

Step by step - use the servis in your code

The servis CLI application just created some python files for the people-api servis. These we need to use in our app:

```
import people_pb2
import people_pb2_grpc
```

We now need to use a GRPC library to interact with the servis for real. What this looks like depends on which programming language you use.

For this workshop, we've provided an example in hello_vis_servis.py. Please rename that file to hello_vis.py (replacing the previous file).

Step by step - add new dependencies

Our Python code is now using additional libraries. If you haven’t already, make sure they are available inside your Docker container.

```
FROM registry.vis.ethz.ch/public/base:charlie

RUN apt install -y python3 python3-pip

COPY requirements.txt /requirements.txt
RUN pip3 install -r /requirements.txt

COPY cinit.yml /etc/cinit.d/demo.yml
ADD src /app
EXPOSE 80
```

Step by step - environment variables

One last piece is missing: To interact with the servis, you need to know on which URL and port it is, and you need an API key.

Your app receives all this information through environment variables, so that you don’t have to hard-code it. That way, it is always up to date.

The variables are:

```
RUNTIME_SERVIS_PEOPLE_API_SERVER
RUNTIME_SERVIS_PEOPLE_API_PORT
RUNTIME_SERVIS_PEOPLE_API_KEY
```

Step by step - add env vars to cinit

The variables are injected in your container by the continuous deployment process, but cinit removes them again for safety reasons!

You need to tell cinit that you want these variables. At the end of your cinit config, add:

```
capabilities:
  - CAP_NET_BIND_SERVICE
env:
  - RUNTIME_SERVIS_PEOPLE_API_SERVER:
  - RUNTIME_SERVIS_PEOPLE_API_PORT:
  - RUNTIME_SERVIS_PEOPLE_API_KEY:
```

Step by step - read the env vars

In Python, you can now read environment variables like so:

```
import os

API_SERVER = os.environ["RUNTIME_SERVIS_PEOPLE_API_SERVER"]
API_PORT = os.environ["RUNTIME_SERVIS_PEOPLE_API_PORT"]
API_KEY = os.environ["RUNTIME_SERVIS_PEOPLE_API_KEY"]
```

Step by step - test it locally

Now, your app isn’t able to run unless these environment variables exist. How can you test it locally?

Don’t worry! visdev got you covered!

If you use visdev, everything will be set up right, and you will even have access to a “fake” people-api for local testing.

Try running the app locally with visdev test and searching for the user “adamif”. Now try searching for your own nethz.

A note on nethz login

The people-api can be used to provide a nethz login page for your application. It has authorization support.

However, to make the login persistent, you will need some extra code (e.g. session management).

We'll leave this for you to figure out during the Hackathon ;)

Questions now?

See you at VIScon!