Idées : <https://www.tutorialworks.com/devops-project-ideas/?utm_content=cmp-true>

* What is an API and how does it work ? <https://www.youtube.com/watch?v=Yzx7ihtCGBs>
* How to Deploy a Flask App to Linux (Apache and WSGI)

<https://www.youtube.com/watch?v=w0QDAg85Oow>

* How to Deploy a Flask Application on Apache | Ubuntu (Basic deployment options)

https://www.youtube.com/watch?v=XeVWtpJHZAU&t=466s

* Build a même Python website (Flask Tutorial for Beginners)

https://www.youtube.com/watch?v=5aYpkLfkgRE

* Python Website Full Tutorial – Flask, Authentification, Databases & More

https://www.youtube.com/watch?v=dam0GPOAvVI

NoteApp : un website qui permet de créer un compte utilisateur + stocker ça dans une base de donnée, se connecter+ se déconnecter + associer des infos à un user spécifique. Une fois qu’on est connecté, on peut créer/supprimer des notes.

Requis :

* pip install flask
* pip install flask-login
* pip install sql-alchemy
* Deploy an application (with high availability) with a database
  + Provision a VM and deploy an application (e.g. WordPress) + a database backend.
  + Figure out how to provide database credentials to the application, through automation.
  + Deploy a second instance of the application, pointing to the same database. Think about high-availability, so if you’re using public cloud, then deploy on 2+ availability zones, or if you’re just working locally then use 2 x VMs.
  + Add a load balancer or reverse proxy to load-balance requests across the two instances, so that you have a single URL to access the application.
  + Store the database credentials in an encrypted form using something like Ansible Vault. When you perform the deployment, fetch the credentials from Vault.
  + Add a cron job which takes backups of the database.
  + Deploy a separate, single instance of the application to a ‘dev’ or ‘test’ environment.
* Create and run a CI/CD pipeline for an app
  + Write a pipeline for your CI/CD tool, to test, compile & package the application. Run the pipeline in the CI/CD tool.
  + Extend the pipeline to deploy it to a server
* Monitoring
  + Configure the application to expose some metrics, to show its health. Explore which metrics the application can expose; e.g. if it’s a web server, can you expose metrics like requests-per-second, or memory usage?
  + Extract or scrape the metrics into a monitoring tool.
  + Create a dashboard to display the metrics in real time.
  + Add monitoring to your underlying infrastructure, too. So if you’re deploying onto virtual machines, then find a way to also monitor disk space, system load, etc.
  + Use a load testing tool to hit your application with extra load. How does it affect the metrics in the dashboard?
* Containerised app
  + Create a pipeline that builds a container image for the application and pushes it to a registry (e.g. Docker Hub, Amazon ECR, self-hosted Nexus container registry)
  + Run the container image with a container engine - e.g. directly on a VM with Docker, or on a container platform like Amazon ECS, Kubernetes…
  + Deploy 2 instances of the container and load-balance between them.
  + Use an orchestration tool like Docker Compose to ensure that the containers are automatically restarted if they are stopped.
  + Try to deploy the containers onto Kubernetes.