Using Kratos and Keto in production

Andrew Minkin (self-employed)

Who am I

Current position: self employed

Last experience:

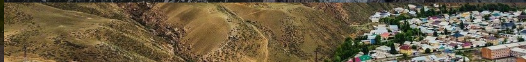
- Go, Python 8+ years
- Operations 7+ years
- Leadership 6+ years

Also:

- Huge fan of DevOps and Open Source cultures

Kyrgyzstan

Photos by
@gladkov_nikolai
@gen1us2k



The story of one re-engineering

The beginning

Internal product for company processes automation

Django monolith

Celery

Path to modernization

Re-host -> Forklift as-is from the data centre

Re-platform -> Forklift with small modifications

Re-factor -> Break up the monolith

Re-engineer -> Serverless, microservice, & awesome

Why we decided to re-engineer current product

- High costs for support
- Bad engineering culture
- A lot of tech debt, small tests coverage
- A lot of own solutions
- Bad security practices
- High cost for infrastructure

Six questions to establish your boundaries

What are your business priorities?

What is the worst possible scenario?

What are your immovable constraints?

What data is this solution storing/processing?

What skills does your team have?

What is the timeline for the project?

https://docs.aws.amazon.com/whitepapers/latest/modern-application-development-on-aws/welcome.html

Our rules for re-engineering

Do not reinvent the wheel

Cost effective hosting for the new product

Use as much as we can from AWS Cloud Provider

ISO 27001 compliant

TDD friendly

Access to production data is strictly prohibited for developers

https://dropbox.tech/infrastructure/rewriting-the-heart-of-our-sync-engine

What are our business priorities

Our main goal is to have idempotent and stable ETL pipelines for data collection

Low time-to-market

Short feedback loop for developers

What is the possible worst case scenario

Data loss -> Wrong salary -> Employee decided to quit the job = Time consuming operation for management and we need to save their time

What are our immovable constraints

ISO 27001

The right to be forgotten

Datasources available through VPN

What data our solution is storing/processing

JIra

- Comments
- Worklogs
- Issue summary
- Changelogs

What data our solution is storing/processing

Gitlab/Github/Bitbucket

- Commits
- Pull(Merge) Requests
- PR’s Commits
- Collaboration in PRs

What data our solution is storing/processing

Future plans

- Slack datasource
- Clickup datasource
- Any other datasource

What skills does our team have

1 TeamLead/Senior Python developer

1 Middle Python Developer

4 Junior Python Developers

1 CTO that has technical vision of the product

What is the timeline for the project

To have production-ready solution in 3 months

Strategy during re-engineering

90% -> new product

10% -> support of current solution

Architecture of new system

SSO/IAM

Airflow for ETL pipelines

SQLAlchemy to work with data

Flask (Because django does not support SQLA)

Metabase as Business Intelligence to build reports

Requirements for SSO/IAM

Easy to use

Self-hosted

Easy to integrate using REST/SDKs

SSO UI for users

Easy to maintain and easy admin UI

Written using team stack (Python/Go/AWS)

What products we were looking at for SSO/IAM

Okta

Auth0

Keycloak

Open source alternatives

Okta

Pros

- Single sign on
- Multi-factor authentication
- Multiple services integration
- Zero-trust security model
- Identity store integration
    a. AD, HR system, etc

Cons

- $2 per user
- No self-hosted solutions

Auth0

Pros

- Single sign on
- API first approach
- Multiple services integration

Cons

- $23 per month. Up to 10k MAU
- No self-hosted solutions

Keycloak

Pros

- Single sign on
- Open source
- Self-hosted

Cons

- Infinispan
- Java

ORY pros

- API first approach
- Good engineering culture
    - Maintainers keep in touch with community
    - Clean code
    - Unit tests
    - Release management

ORY pros

Zero trust model

MFA (in future)

ORY products associations

ORY products associations

Why we chose ory

Written in Go

Good engineering culture

Contributors are open to the community

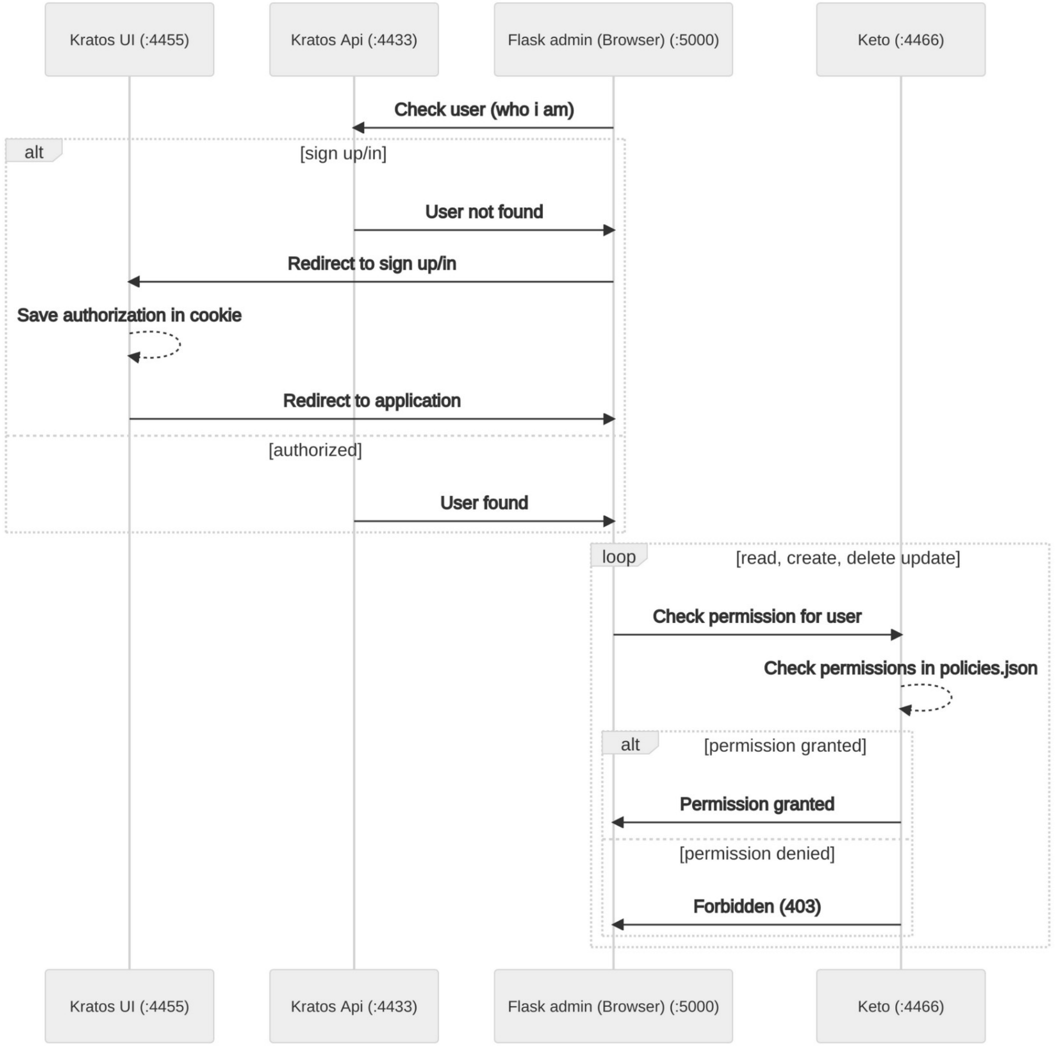
- Meetups
- Communication in github issues

What else we took from ory

SQA page

A few tips to write documentation

IAM flow


Kratos UI (:4455)
Kratos Api (:4433)
Flask admin (Browser) (:5000)
Keto (:4466)
Check user (who i am)
alt
[sign up/in]
User not found
Redirect to sign up/in
Save authorization in cookie
Redirect to application
[authorized]
User found
loop
[read, create, delete update]
Check permission for user
Check permissions in policies.json
alt
[permission granted]
Permission granted
[permission denied]
Forbidden (403)
Kratos UI (:4455)
Kratos Api (:4433)
Flask admin (Browser) (:5000)
Keto (:4466)

Integrating kratos to our solution

REST API

Python package

- Middleware in flask
- Decorators

Integrating keto to our solution

CI/CD pipeline for keto

- Keto permissions migrate
- Managing policies using keto_policies folder in the gitlab repository

Deploying to production

AWS/EKS/RDS

https://github.com/maddevsio/aws-eks-base

Terraform/k8s/Helmcharts

Key takeaways

Keto & kratos are ready for SMB

We saved

3*140*6=2520 man-hours (minimum $40k)

Spent 40 hours to integration

ORY is a great example of excellent engineering culture

Questions?

Inst: @Gen1us2k

LinkedIn: Andrew Minkin

Github: @Gen1us2k