#INFRA PLANEJAMENTO MIGRAÇÃO CKAN

---

1. **Motivação**:
    a. Volume elevado de recursos da equipe de infra é gasto para resolver bugs e issues na infraestrutura do CKAN atual
    b. CKAN exige que se crie muitos plugins para coisas que gostaríamos que fossem nativos.
    c. Temos uma série de desejos de features que o CKAN é engessado para permitir.
    d. Gostaríamos de ter uma estrutura de permissão mais sofisticada. A estrutura de permissão do CKAN não atende às nossas necessidades.
2. **Contra-argumento**:
    a. Mesmo após a migração, será preciso um investimento de tempo elevado para manutenção da nova infraestrutura de metadados.
3. **Features - Não negociáveis:**
    a. Pesquisa com filtros/slices/contagens (Solr/ElasticSearch)
    b. Gerenciar packages e resources (CRUD)
        i. Create: Validação (Pydantic) / Schema para o frontend montar o form
        ii. Update: Validação / Schema / Atualizar o search index / Patch/Put / (audit log?)
        iii. View:
        iv. Delete: (audit log?)
    c. Gerenciar Usuários: hoje temos basicamente só usuário normal e admin.
    d. Backup do banco e dos binários
    e. Schema para entidades fundamentais: usuários, organizações, temas, etiquetas, conjuntos, fontes originais, pedidos LAI, tabelas BD+
    f. Endpoints já existentes no website https://basedosdados.org/openapi
    g. Usar terraform + kubernetes
    h. Ambiente dockerizado
    i. Ambiente de teste local/cloud:
        i. Framework de testes (conexão e mock de banco e outros serviços já pronto pro dev poder adicionar testes com pouco esforço)
    j. Documentacao e Onboarding.
    k. HTTPS e renovação de certificado

4. **Features - Desejáveis:**
    a. Deploy: Fazer um push e o deploy acontecer magicamente.
    b. Infrastructure as a service: terraform, IAM
    c. Schema para novas entidades fundamentais: documentos, anexos, análises comunitárias (GitHub)
    d. Internacionalização
    e. Desacoplar schema de metadados
        i. Permitir que organizações personalizem seus schemas.
        ii. Permitir que a validação de metadados não dependa do servidor da BD estar online.
    f. Potencializar espaço de usuário: fórum, seguir bases, subir dados, postar análises, conectar com serviços (GCP, AWS, Kaggle), etc.
    g. Web 2.0 - Comentários / Likes / Salvar buscas / Notificação de atualização / Badges / Quem baixou
    h. Estruturar IAM (em acordo com GCP e sistema de permissões na BD)
    i. Permitir abstração de nomes de conjuntos e tabelas materializadas em cima de tabelas usando IDs/hash como nome. GitHub Issue #425
    j. Permitir sistema de gestão de dados e metadados direto via web.
        i. Ambiente de desenvolvimento (back-end prefect)
        ii. Inferência de metadados
        iii. Validação de metadados
        iv. Data checks
    k. Estrutura para Metadata Graph
    l. Separar Front do Back (também no repositório GitHub. Hoje é mono repo `website`)
5. **Stack**
    a. Back-end
        i. Django
        ii. FastAPI
    b. Banco de dados do back-end
        i. Postgres
    c. ORM
        i. Sqlalchemy
        ii. SQLmodel
        iii. Django ORM
    d. DB migrations
        i. Alembic
        ii. Django ORM
    e. Search
        i. Postgres FTS

        ii. Elastic Search
        iii. Solr
6. **Plano de ação**
    a. Mapear custos e benefícios de opções de stack: features, custo de migração, manutenção, comunidade, etc.
    b. Decidir stack.
    c. Mapear pontos de mudança no nosso ecossistema.
    d. Design do banco de dados (entidades, relacionamentos)
        i. Diagrama lógico entidade/relacionamento
    e. Desenhar a interface publica do backend (endpoints/pushs?)
    f. Montar MVP em cluster paralelo.

Status prefeitura:

Usavam planilhas de arquitetura para preenchimento de metadados. Problema: ficava bagunçada com muitas pessoas diferentes fazendo. Solução foi criar uma API com Django. O maior problema inicialmente foi a criação de uma interface. Usou-se o Django Admin para lidar com isso.

Para segunda versão tentou-se criar uma interface mais amigável.

Stack:

1. API: FastAPI
2. Cache: redis (estudar como integrar com FastAPI) (https://developer.redis.com/develop/python/fastapi/)
3. Postgres
4. ORM: SQLmodel
5. ElasticSearch (estudar se é a melhor opção)
6. DB migrations: Alembic

Primeiros passos:

1. Usar repositório API que já existe na Organização da BD
2. Replicar os endpoints que temos atualmente usando FastAPI (estrutura e arquitetura do repositório será criada pela equipe da prefeitura)
3. Separar FrontEnd de BackEnd
4. Criar design do banco de dados

Obs: 2 e 4 irão ocorrer em paralelo