URL Shortener (Project Planning Task)

Table of Contents

Project Scope and Limitations

1. We want to launch an MVP as soon as possible.
2. Since the project is at the idea-evaluation stage, we want to spend as minimum resources as possible. This includes both an initial R&D capital investment and operational costs for the infrastructure and ongoing support. I.e. minimize the TCO.
3. Uncertainty:
   a. Do we want to collect any telemetry and sell it afterward?
   b. Do we want to add ad-monetization to the service? Maybe a paid option to create an ad-free links for the URL-creators?
   c. Are there any government regulations we need to meet?
   d. For the purpose of the further planing it is assumed that all of those points are unclear and should not be brought into the scope of MVP.
4. We don’t need a session or account support, meaning there will be no way to see all the URLs you’ve shortened so far and manage them. It is enough if the URL-creator will be able to access his URL-management page right after shortening or later if they save/bookmark that management page.

High-Level Design

Serverless

Suitable for idea-evaluation with MVP phase

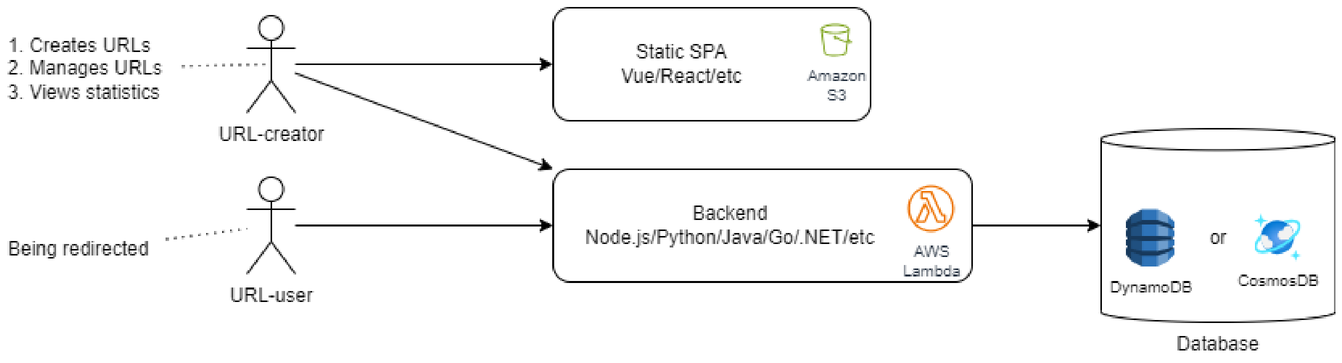

Figure 1. Serverless Architecture Diagram

Pros:

1. Negligible setup costs.
2. Negligible constant costs.
3. Negligible maintenance burden.
4. Out of the box scalability, availability, logging, monitoring, and other perks.

Cons:

1. Possible vendor lock-in if don't take precaution separating our infrastructure access layer with decent abstractions.
2. Less attractive costs per click at high load compared to a dense compute resources.

Cloud-Native

Suitable for large scale phase or if the company already has some infrastructure and it is available to allocate some capacity out of it.

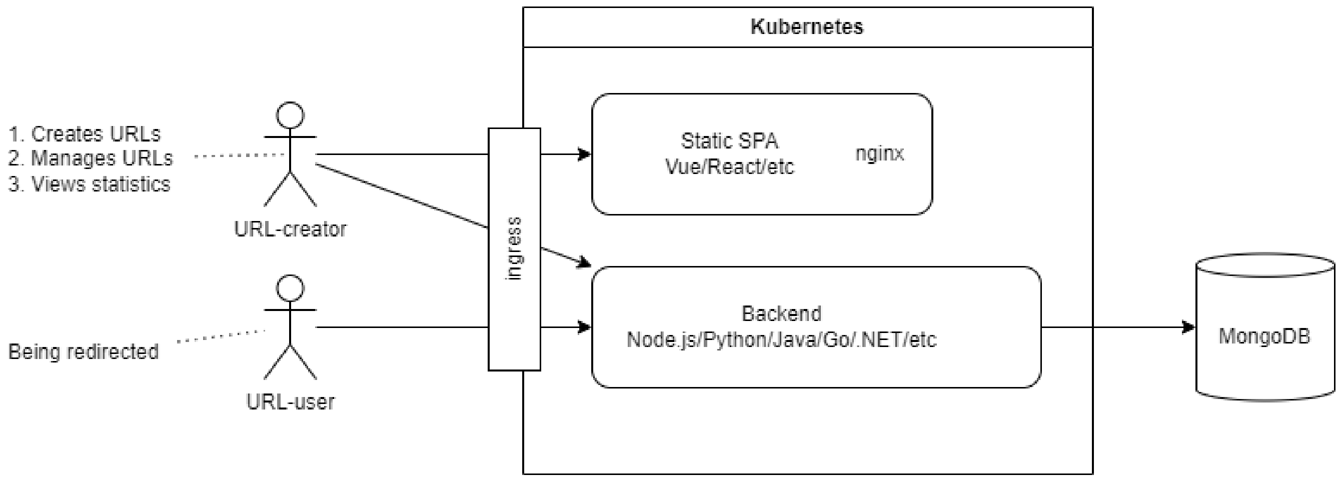

Figure 2. Cloud-Native Architecture Diagram

Pros:

1. Vendor independence (Amazon/Azure/GCP/On-prem/etc.)
2. Lower costs per click at high volumes.
3. Fine-grained control over the solution.

Cons:

1. Higher constant costs.
2. High initial setup costs.
3. Ongoing maintenance burden.
4. Scalability, availability, logging, monitoring, and other perks are up to our choice of configuration leading to a corresponding time, resources and attention investment.

Key Technical Decisions

1. We’ll store the management token inside the URL.
2. We’ll store the statistics counter inside the URL database entry.
3. We’ll update the counter using optimistic compare-and-swap algorithm which should be suitable for the 1000 clicks per day load given in the requirements.

Required Resources

People:

1. 1 DevOps
2. 1-2 Full-stack engineers **or**
    a. 1 Back-end engineer
    b. 1 Front-end engineer
3. 1 QA

Resources:

1. Git-Hub Subscription
2. AWS Subscription

Technologies

The product is pretty simple and can be implemented using any modern front-end and back-end frameworks.

Given that we want to launch as soon as possible, the most rational choice is to stick to the technologies the team has experience in.

For Oktopost I believe it is the following:

1. Front-end: `Vue.js`
2. Back-end: `PHP` (+ some framework like `Laravel` or whatever the team is familiar with)
3. Database: `MongoDB` (`CosmosDB` with `MongoDB` protocol for serverless)
4. Infrastructure:
    - Serverless option:
        a. Amazon S3
        b. Amazon API Gateway
        c. AWS Lambda
        d. CosmosDB
    - Cloud-native option:
        a. Amazon EKS
        b. MongoDB

Security Considerations

The only identified risk is an API abuse causing high cloud charges.

To mitigate this risk we'll configure a rate limit at the Amazon API Gateway level.

Tasks Breakdown

No	Phase	Responsible	Task	Estimation (days)
P1	Planning	Team-Lead + Product Owner	Product scope clarification, clearing up ambiguities and uncertainties	1
P2	Planning	Team-Lead	Tasks Specification	1
P3	Planning	Team	Project kick-off and team alignment	1
D1	Development	DevOps	Development environment provisioning and CI/CD configuration	2
D2	Development	Front-end (or Full-stack)	URL creation page development	1
D3	Development	Front-end (or Full-stack)	URL management (statistics) page development	1

No	Phase	Responsible	Task	Estimation (days)
D4	Development	Back-end (or Full-stack)	URL creation endpoint development	1
D5	Development	Back-end (or Full-stack)	URL redirection endpoint development	1
D6	Development	Back-end (or Full-stack)	URL info endpoint development	1
D7	Development	Back-end (or Full-stack)	URL management (update) endpoint development	1
Q1	QA	DevOps	QA environment provisioning and CI/CD configuration	2
Q2	QA	QA	Test Plan specification	1
Q3	QA	QA	Testing	2
Q4	QA	Front-end/Back-end/Full-stack	Fixing bugs	1
R1	Release	DevOps	Production environment provisioning and CI/CD configuration	3
R2	Release	Product Owner	Release Approval	1

Timeline

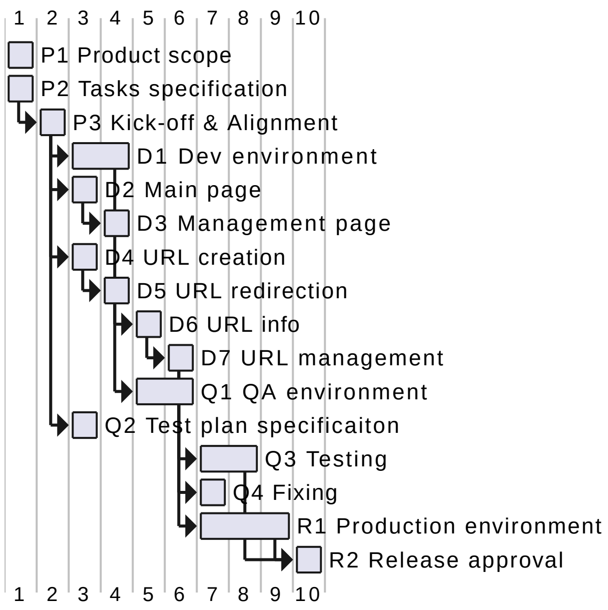

Figure 3. Project timeline

Total: 10 days (2 weeks) + around 20-30% risk.