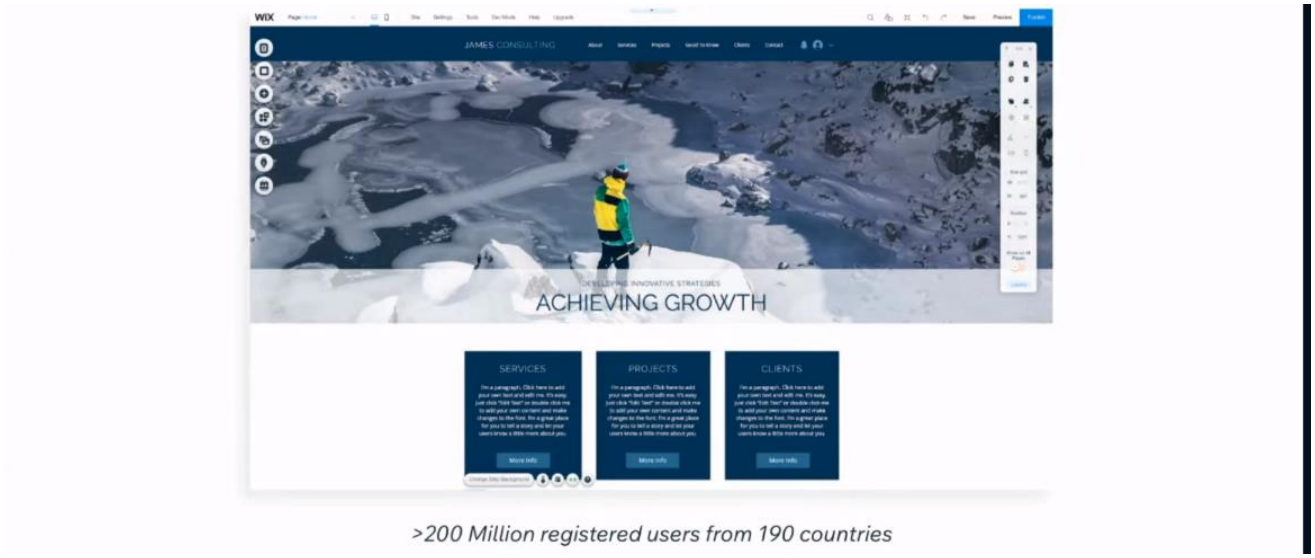
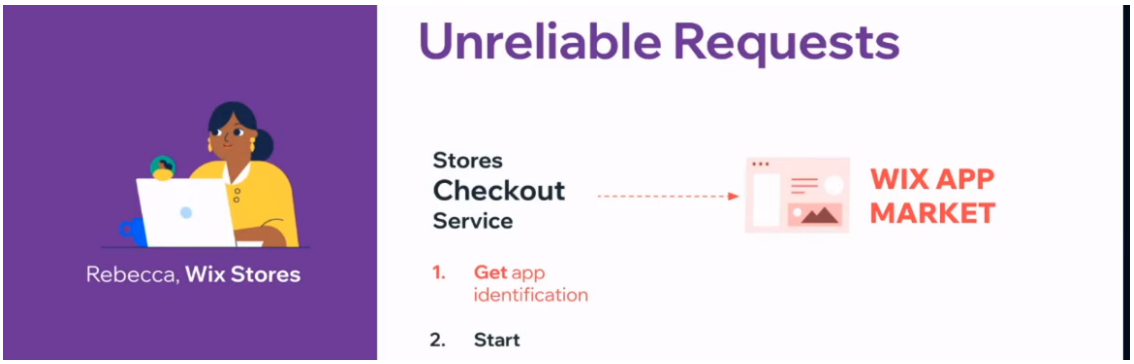
Wix has a huge scale of traffic. More than 500 billion HTTP requests and more than 1.5 billion Kafka business events per day. This talk goes through 4 Caching Patterns that are used by Wix's 1500 microservices in order to provide the best experience for Wix users along with saving costs and increasing availability. A cache will reduce latency, by avoiding the need of a costly query to a DB, a HTTP request to a Wix servicer, or a 3rd-party service. It will reduce the needed scale to service these costly requests. It will also improve reliability, by making sure some data can be returned even if aforementioned DB or 3rd-party service are currently unavailable. The patterns include:

- Configuration Data Cache - persisted locally or to S3
- HTTP Reverse Proxy Caching - using Varnish Cache
- Kafka topic based 0-latency Cache - utilizing compact logs
- (Dynamo)DB+CDC based Cache and more - for unlimited capacity with continuously updating LRU cache on top.

Each pattern is optimal for other use cases, but all allow to reduce costs and gain performance and resilience.

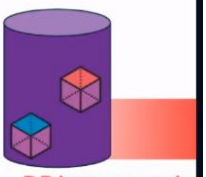




Alex, **Wix Stores**

Request Overload

Stores Catalog Service → Store Catalog



DB latency and failure increase

No Cache - Increase in

- loading and response
- Network costs
- DB failure

With Cache -

- reduce latency
- reduce the needed scale
- improve reliability

When & how to Cache

Examples from 2000 microservices of various use cases

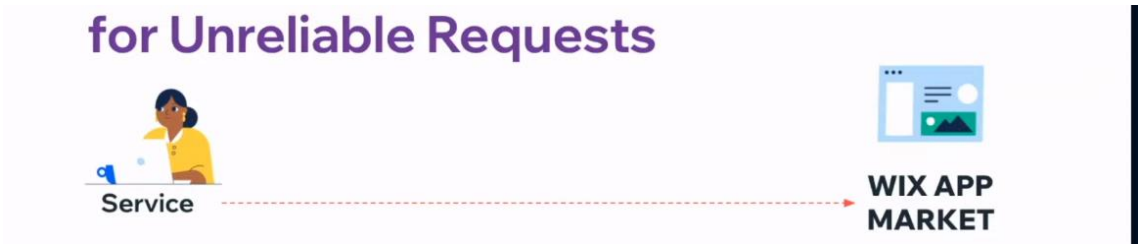
#1 high risk/cost of network failure	#2 To improve average latency for data layer access	#3 Some very high external traffic cases	#4 When NOT to - young products
Like caching external critical configuration data.	DynamoDB + CDC + LRU cache.	Caches are very important - cache before the service.	Don't cache prematurely

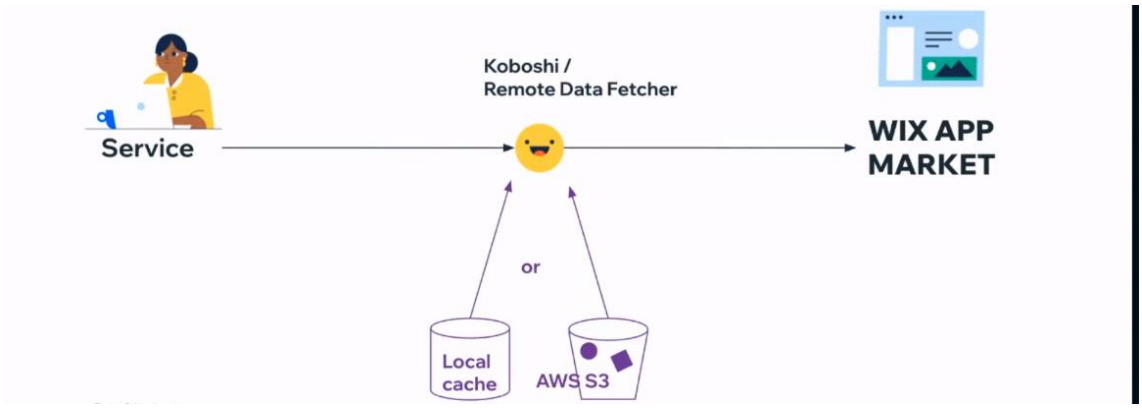
Agenda

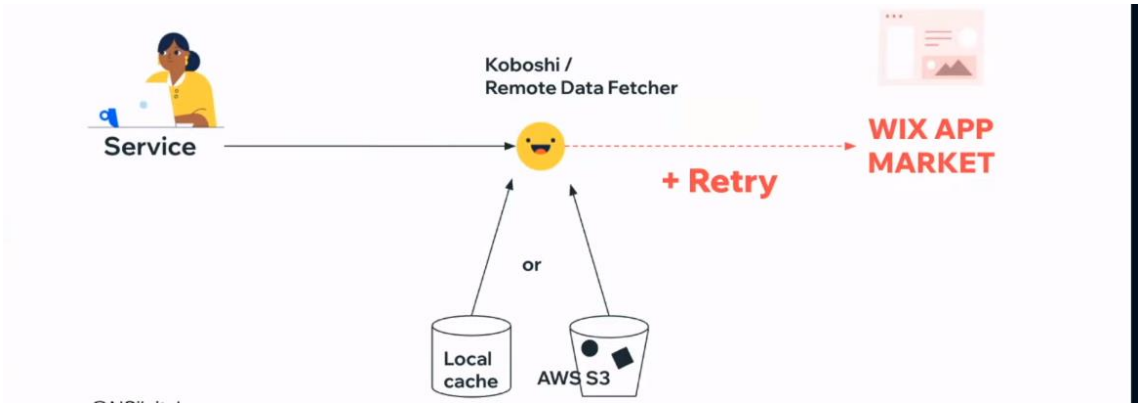
3 Caching Patterns:

1. S3 backed static cache
2. DynamoDB+CDC based cache
3. HTTP Reverse Proxy cache

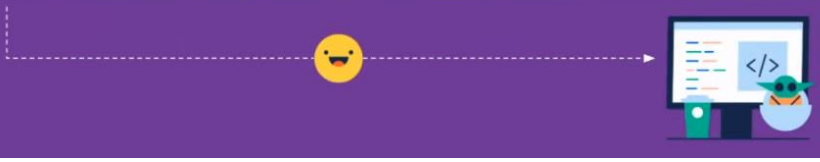
for Unreliable Requests







Cache Type	Cache Size	Best For	Not For
Read-through cache **Before** the 3rd party service	Data has to fit into memory	External source of static configuration	Dynamically updating data

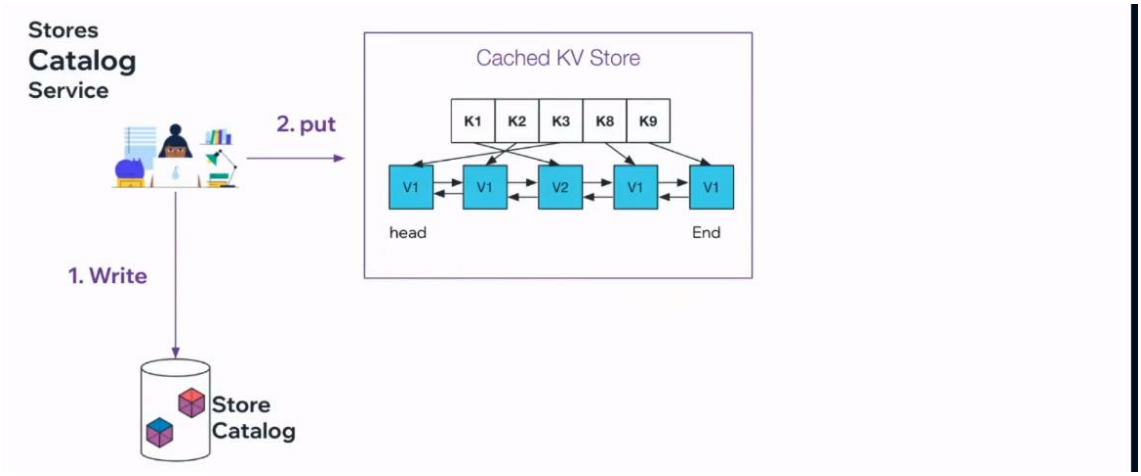


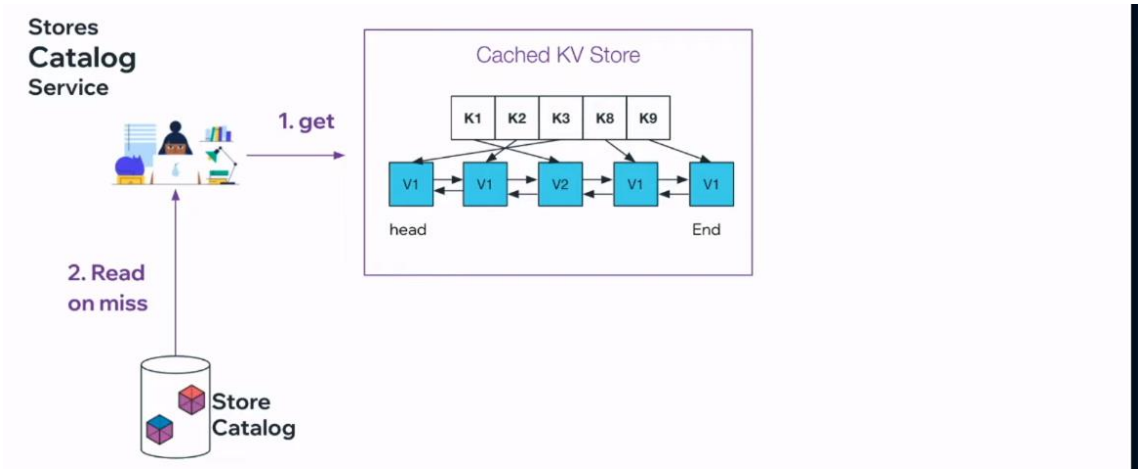
Next

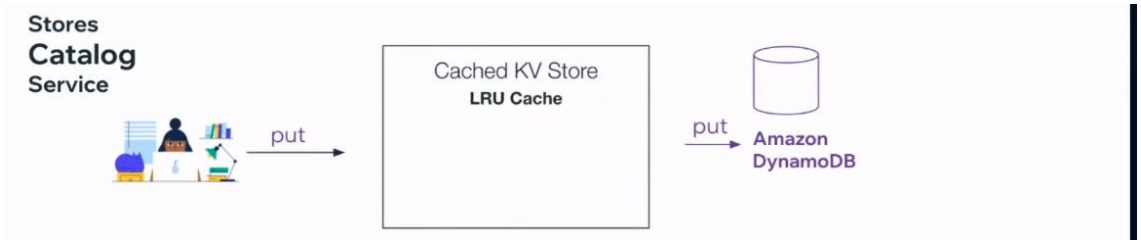
3 Caching Patterns:

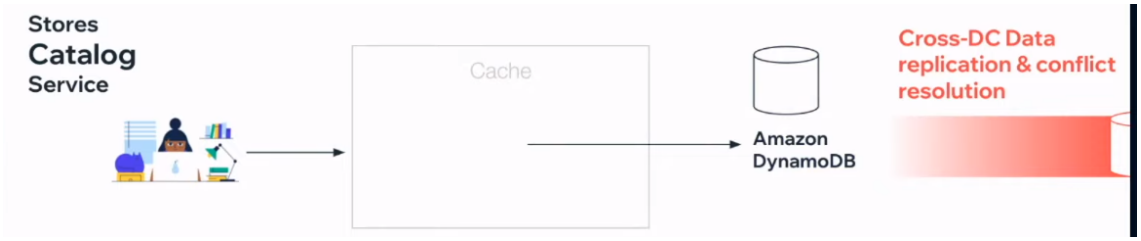
1. S3 backed static cache
2. **DynamoDB+CDC based cache**
3. HTTP Reverse Proxy cache

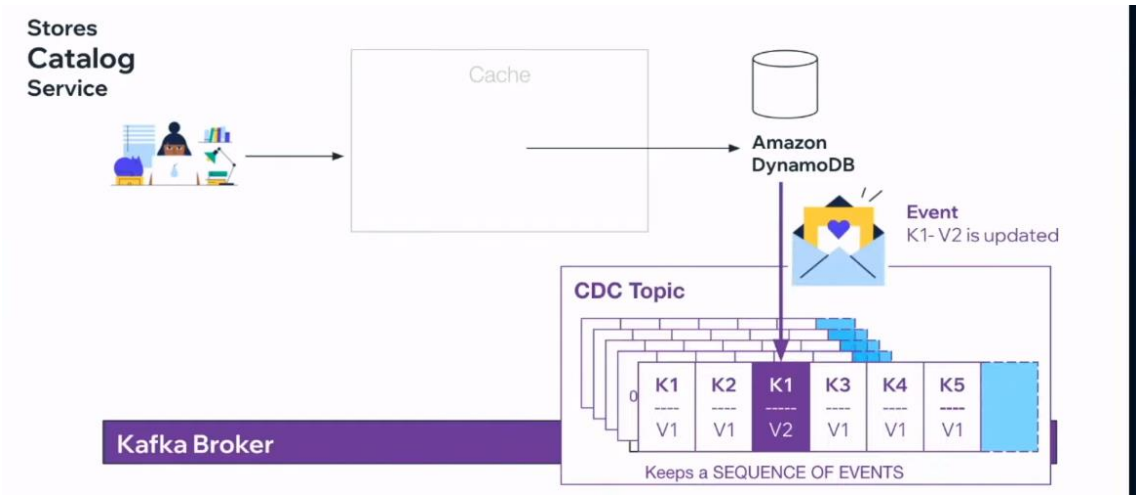
for Request Overload

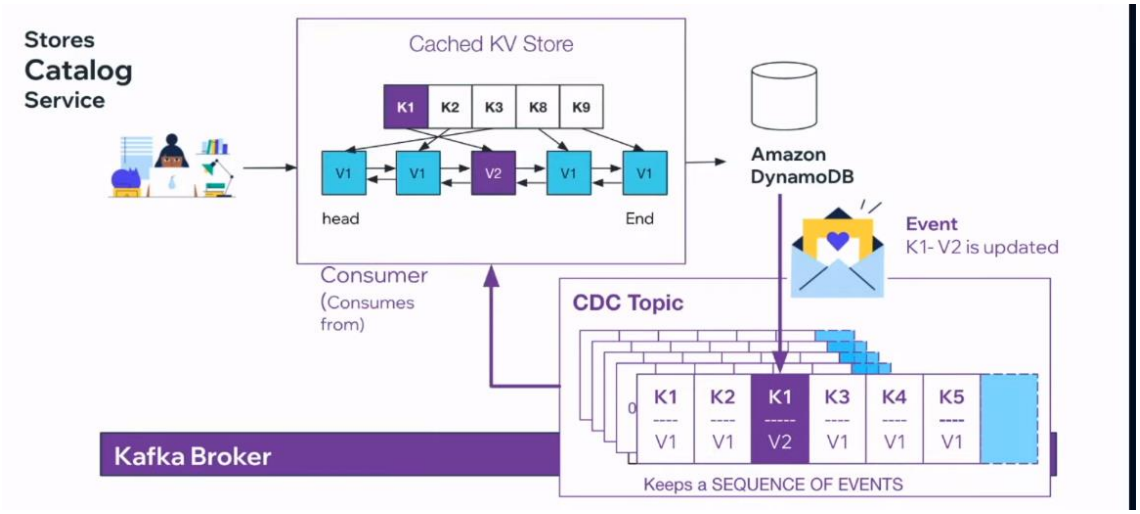


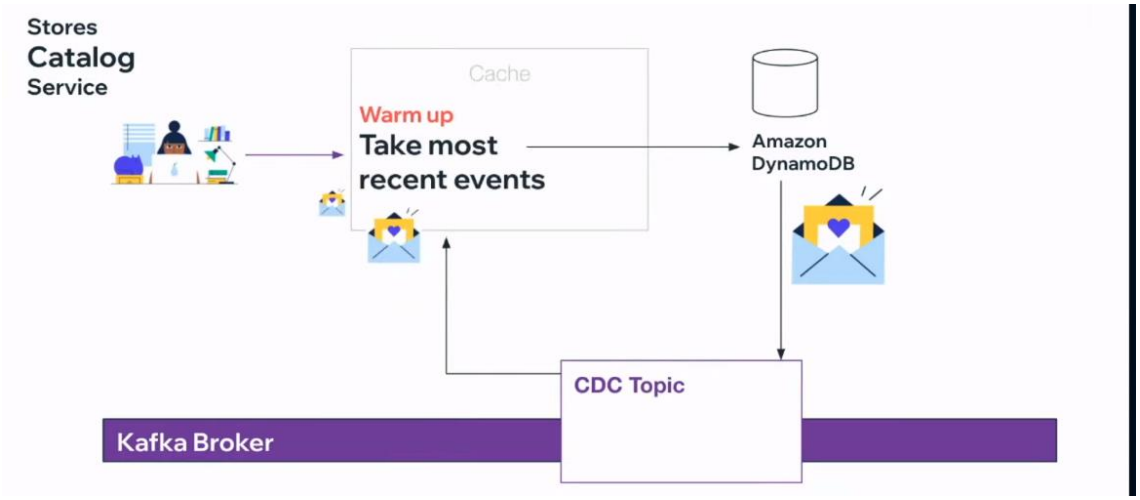


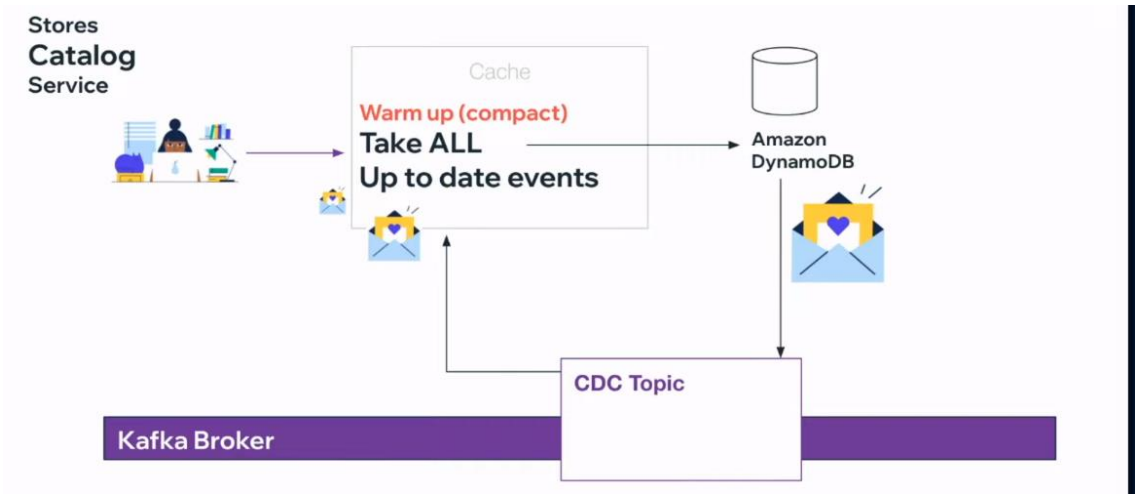




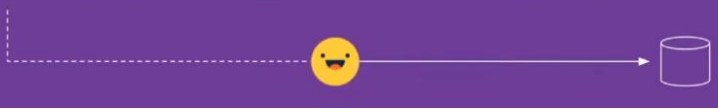

Stores
Catalog
Service
Cache
Amazon
DynamoDB
Event
K1- V2 is updated
CDC Topic
K1 K2 K1 K3 K4 K5
V1 V1 V2 V1 V1 V1
Kafka Broker
Keeps a SEQUENCE OF EVENTS


Stores
Catalog
Service
Cached KV Store
K1 K2 K3 K8 K9
V1 V1 V2 V1 V1
head
End
Consumer
(Consumes
from)
Amazon
DynamoDB
Event
K1- V2 is updated
CDC Topic
K1 K2 K1 K3 K4 K5
V1 V1 V2 V1 V1 V1
Kafka Broker
Keeps a SEQUENCE OF EVENTS


Stores
Catalog
Service
Cache
Warm up
Take most
recent events
Amazon
DynamoDB
CDC Topic
Kafka Broker



Cache Type	Cache Size	Best For	Not For
Cache-Aside **Before** the DB	No limitation on DB data size LRU cache size is configurable	Make DB read access faster for "popular" values	Highly critical application startup information



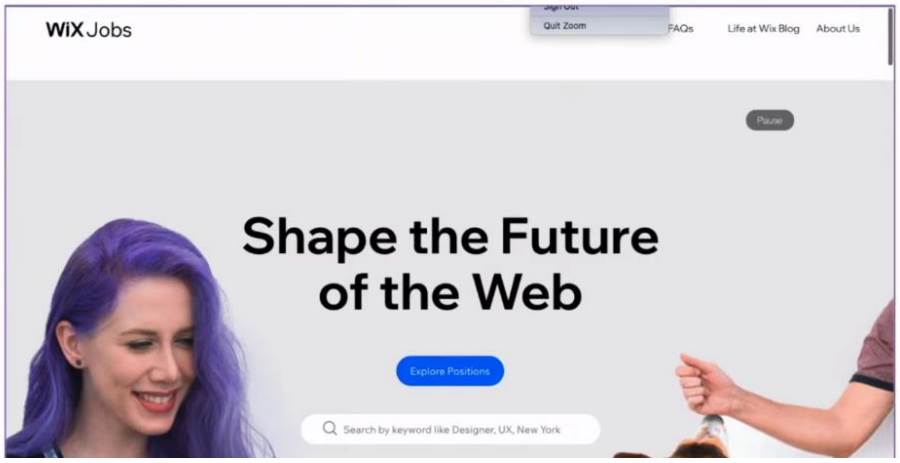
Next

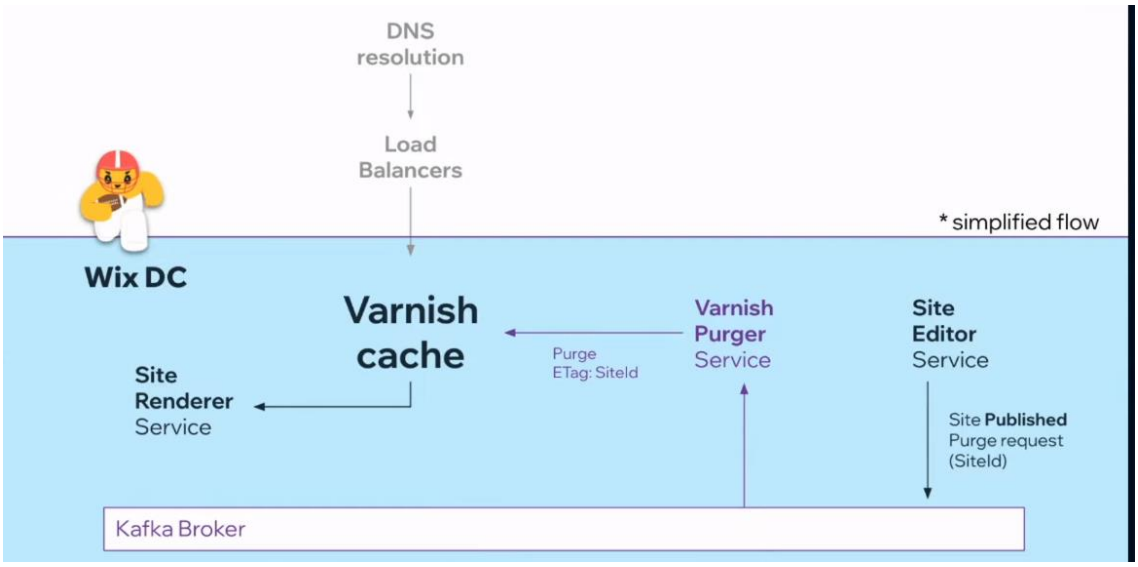
3 Caching Patterns:

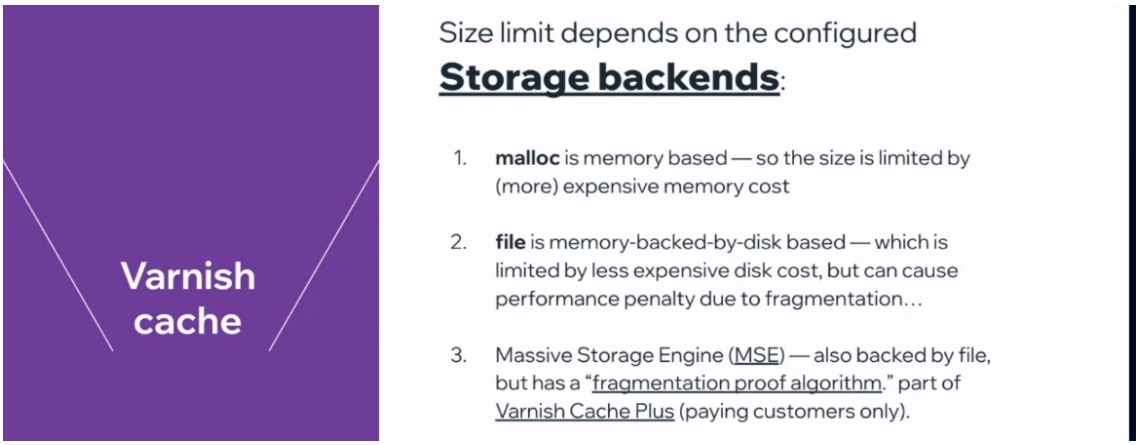
S3 backed static cache

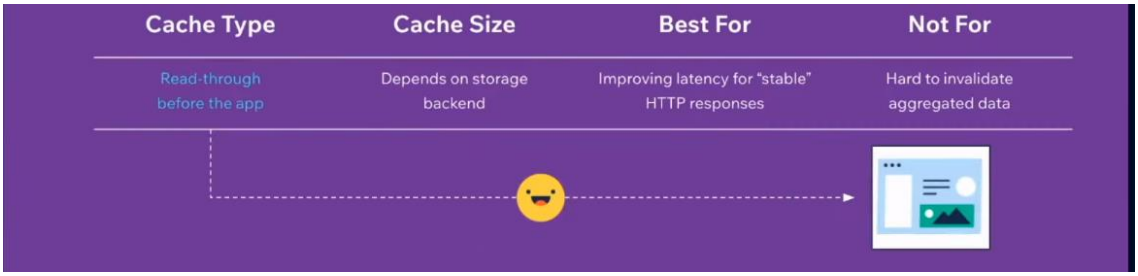
DynamoDB+CDC based cache

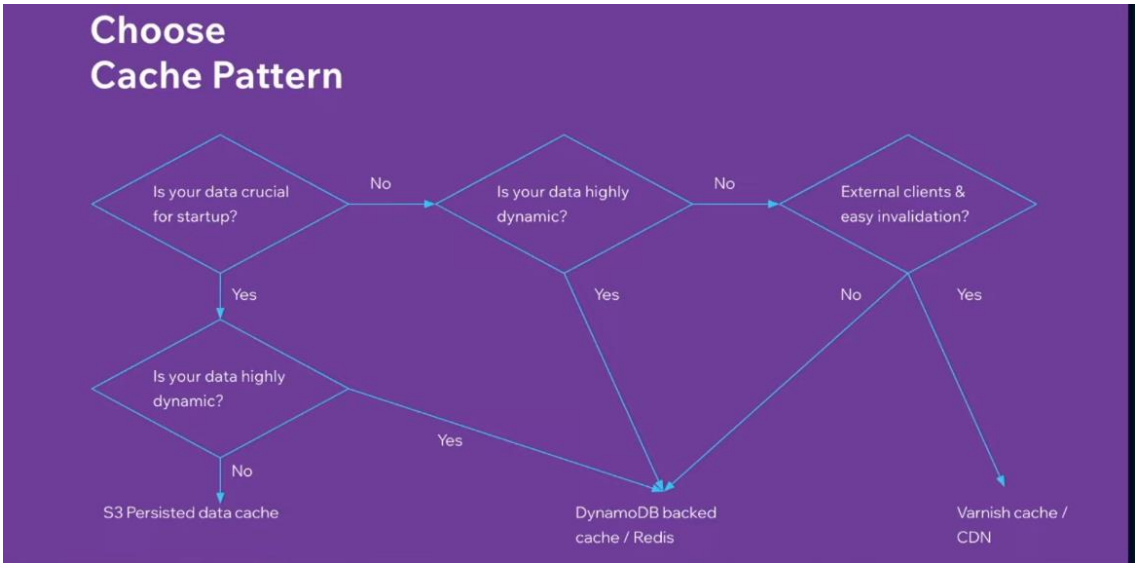
HTTP Reverse Proxy cache




DNS resolution
Load Balancers
* simplified flow
Wix DC
Varnish cache
Varnish Purger Service
Purge ETag: SiteId
Site Editor Service
Site Renderer Service
Site Published Purge request (SiteId)
Kafka Broker


Varnish cache
Size limit depends on the configured
Storage backends:
1. malloc is memory based — so the size is limited by (more) expensive memory cost
2. file is memory-backed-by-disk based — which is limited by less expensive disk cost, but can cause performance penalty due to fragmentation…
3. Massive Storage Engine (MSE) — also backed by file, but has a "fragmentation proof algorithm." part of Varnish Cache Plus (paying customers only).

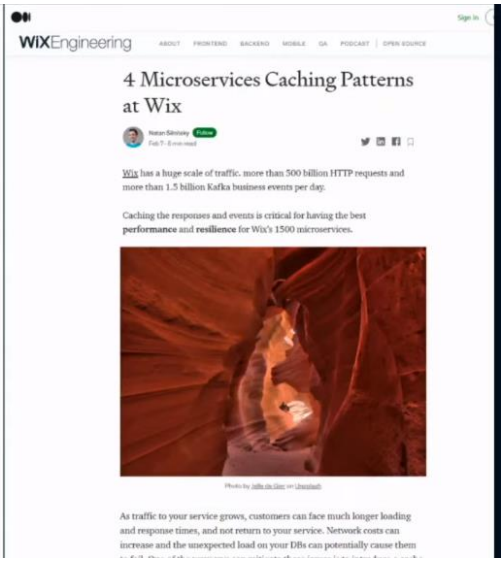

Cache Type
Cache Size
Best For
Not For
Read-through before the app
Depends on storage backend
Improving latency for "stable" HTTP responses
Hard to invalidate aggregated data


Choose Cache Pattern
Is your data crucial for startup?
No
Is your data highly dynamic?
No
External clients & easy invalidation?
Yes
Yes
No
Yes
Is your data highly dynamic?
Yes
No
S3 Persisted data cache
DynamoDB backed cache / Redis
Varnish cache / CDN

The blog post

https://medium.com/wix-engineering/4-microservices-caching-patterns-at-wix-b4dfee1ae22f

@NSilnitsky



github.com/wix/greyhound

A Java/Scala high-level SDK for Apache Kafka.

0.2 is out!