SANTA'S TECH STACK
Situation
<table>
<thead>
<tr>
<th>Requirement</th>
<th>eCommerce</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track naughty &amp; nice behaviors.</td>
<td>User Profiles</td>
</tr>
<tr>
<td>Allow people to send in a list of toys.</td>
<td>Wishlists</td>
</tr>
<tr>
<td>Have a list possible of toys and gifts</td>
<td>Product Information Management (PIM)</td>
</tr>
<tr>
<td>Keep track of how many gifts have been built by the elves</td>
<td>Inventory</td>
</tr>
<tr>
<td>Keep track of deliveries to ensure everyone gets either a gift or coal</td>
<td>Carts / Orders</td>
</tr>
<tr>
<td>Make sure when using the system people experience the joy of the holidays.</td>
<td>Unique front-ends (website, mobile, etc.)</td>
</tr>
</tbody>
</table>
Large Scale System

- 8 Billion users
- Yearly activity, but surge in data collection during December
- All fulfillment happens in a single night. (requires high-throughput)
- Users across the world. (global distributed system)
- Every possible language. (multi-language)
- Security Compliance (GDPR, ADPPA, COPPA, etc.)
- Full accessibility (a11y)
Role of Architect

• Have a dedicated architect.
• Architecture decisions will impact the entire project and development.
• Choosing the wrong design can create lock-in, an inability to scale, or project failure.
• Be sure the architect is involved at the beginning of the process with Business Owners and Product Managers.
Scaling
2 Trips
Vertical Scaling

- Easiest and most common way to scale.
- Adding better hardware or for the cloud increasing capacity by adding more processing power.
- Increase storage, CPU, memory, and other resources.
- May be the best option for specific tasks (i.e. CPU bound calculations)
- Even with the cloud, vertical scaling has clear limits.
Horizontal Scaling

- Increased complexity and costs.
- Ability to auto-scale by adding additional servers, containers, etc.
- Can be layered on to the existing system, less downtime.
- Built-in high availability and fault tolerance.
- May be the best option for some tasks (i.e. Network Traffic)
Just use the cloud.

It’s not as simple as adopting cloud specific options:
- Cloud offerings often create lock-in.
- We may need redundancy between clouds.
- Data residency laws may prohibit certain cloud options.

Instead, we should focus on using industry standard tools. These gravitate towards open-source.
Partitioning

- A load balancer can be leveraged to send traffic accordingly.
- For databases either sharding or partitioning must be used.
- Consider logical partitions based on business rules:
  - Data residency laws
  - Dedicated databases for specific functions
Breaking things up.
• User Profile
• Wish List
• Product Information
• Search
• Order Information
• Inventory
• Etc.
Each team:

• Full ownership & autonomy
• You build it, you run it
• Focused innovation
Microservices
A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and should implement a single business capability.
Characteristics of a Microservice

- Services are small, independent, and loosely coupled.
- Each service is a separate codebase.
- Services can be deployed independently.
- Services are responsible for persisting their own data or external state.
- Services communicate with each other by using well-defined APIs.
- Services don’t need to share the same technology stack, libraries, or frameworks.
- Owned by a small team.
Assuming it’s just tech

Team structure is equally important.
Sharing a database

This creates a single dependency
Layered Service Architecture

Be sure teams can work and deploy independently of other teams.
Lack of Internal Versioning

Each microservice should have versioning.
DevOps, CI/CD, Monitoring, Container Orchestration become more important.
Technology
Services

CLIENT

User Service

Products Service

Orders Service

k8s
Frontend / Client
Traditional Server-Rendered Website vs Single-Page Application (SPA)
Image Service (DAM)

Multi-Vibe

$325.00 IN STOCK

Inventory Service

Product Service

Description

Abstract geometric lampshade and matching base creates a crisp modern-looking piece of art, rather than a typical boring lamp. Designed especially for those who love to stand out and shine.
Backend For Frontend

- User Service
- Products Service
- Orders Service

Transitional App

BFF

k8s
GRAPHQL AS AN AGGREGATION SERVICE

GraphQl as Aggregation Service

- Web App
- Mobile App
- Other

- Commerce
- Taxes
- Search
- Payment
- Identity
- Shipping
- Marketing
GraphQL Pain Points

What are your main pain points when using GraphQL?

- Error Handling: Round 1 wins 803, Round 2 wins 447, Round 3 wins 241
- Performance: Round 1 wins 738, Round 2 wins 433, Round 3 wins 244
- Client-side Caching: Round 1 wins 694, Round 2 wins 365, Round 3 wins 206
- Security: Round 1 wins 640, Round 2 wins 318, Round 3 wins 153
- API Versioning: Round 1 wins 656, Round 2 wins 290, Round 3 wins 140
- File Upload: Round 1 wins 618, Round 2 wins 290, Round 3 wins 140
- Combining Different ...: Round 1 wins 574, Round 2 wins 249, Round 3 wins 140
- Testing: Round 1 wins 559, Round 2 wins 246, Round 3 wins 140

User Count
GraphQL Benefits

What are the main reasons why you enjoy using GraphQL?

- Type-checking
- Introspection & API
- Avoiding Over-fetching
- Aggregating Requests
- Tooling & Ecosystem
- Combining Different
- Fragment Colocation
- Community

User Count
Enter Open API Spec (aka Swagger)

- Writing the spec first ensures proper API design and an API-first approach.
- Specification provides a document similar to introspection, outlining the REST interface.
- Creates better tooling by generating Docs, SDKs, and more.
- Provides better versioning by tracking changes across all services.
Services

Transitional App

BFF

User Service

Products Service

Orders Service

k8s
Services

- Transitional App
- BFF
  - User Service
  - Products Service
  - Orders Service
- k8s
Services

- Transitional App

BFF

User Service

Products Service

Orders Service

k8s
API Routes

Examples

- Basic API Routes
- API Routes with GraphQL
- API Routes with REST
- API Routes with CORS

API routes provide a solution to build your API with Next.js.

Any file inside the folder `pages/api` is mapped to `/api/*` and will be treated as an API endpoint instead of a page. They are server-side only bundles and won't increase your
The serverMiddleware property

Define server-side middleware.

- Type: Array
  - Items: String or Object or Function

Nuxt internally creates a connect instance that you can add your own custom middleware to. This allows us to register additional routes (typically /api routes) without need for an external server.

This allows you to create a client API/server API pattern using Nuxt alone. This means that from the browser (for example, within a Vue component) you can make a request to a route in your server middleware.

One benefit of this pattern is that the server middleware exists on the server (like most middleware), not on the client. This means that you can handle environment variables and secrets in the server middleware, without exposing that information to the user.

Because connect itself is a middleware, registered middleware will work with both nuxt start and also when used as a middleware with programmatic usages like express-template. Nuxt Modules can also provide serverMiddleware using this.addServerMiddleware()
+server

As well as pages, you can define routes with a +server.js file (sometimes referred to as an 'API route' or an 'endpoint'), which gives you full control over the response. Your +server.js file (or +server.ts) exports functions corresponding to HTTP verbs like GET, POST, PATCH, PUT and DELETE that take a RequestEvent argument and return a Response object.

For example we could create an /api/random-number route with a GET handler:

```javascript
import { error } from '@sveltejs/kit';

/** @type {import('./$types').RequestHandler} */
export function GET({ url }) {
  const min = Number(url.searchParams.get('min') ?? '0');
  const max = Number(url.searchParams.get('max') ?? '1');

  const d = max - min;

  if (isNaN(d) || d < 0) {
    throw error(400, 'min and max must be numbers, and min must be less than max.'
  }

  const random = min + Math.random() * d;
```

https://kit.svelte.dev/docs/routing#server
API SaaS
Services

- Transitional App
  - BFF
- User Service
- Products Service
- Orders Service
- k8s
Services

Transitional App

- BFF

- strapi
- builder.io
- elasticsearch
- algolia
- elasticpath
Thank you!
CONNECT

James.Luterek@elasticpath.com
In/jamesluterek
/jluterek
/jamesluterek