Platform-Powered 🚀

Build a frontend platform that scales as fast as you do

Andrew Hao (@andrewhao)

Ever felt growing pains?

"...most tools and processes only support about one order of magnitude of growth before becoming ineffective"

Will Larson. *An Elegant Puzzle: Systems of Engineering Management*

Monolith to Microservices

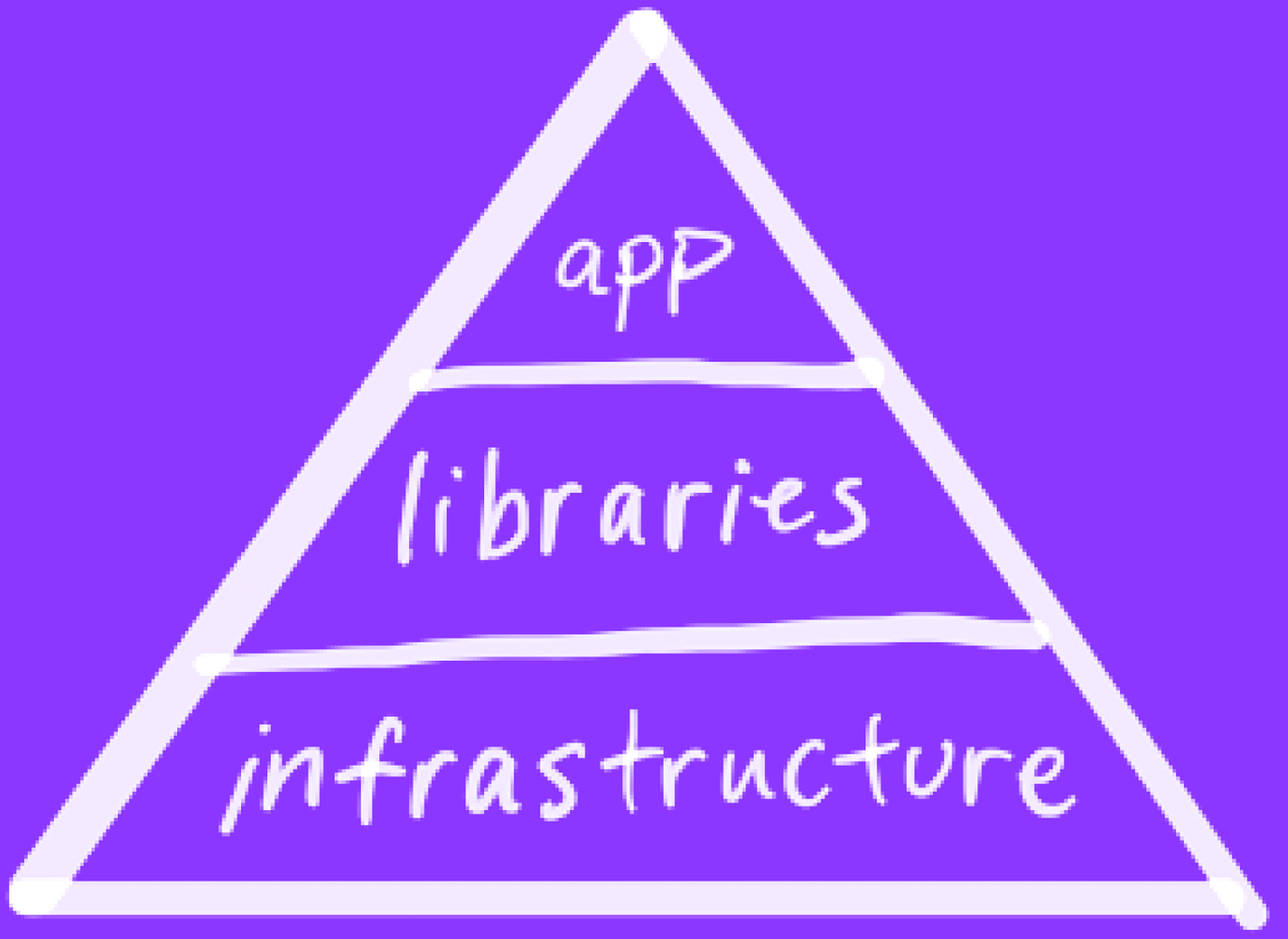
- Python/Angular monolith
- ...to Node + React isomorphic apps via a templated service generator
- ...coupled with infrastructure investments in microservices
- ...led to a microservice explosion! 💥

But the problems were starting to catch up to us

- Long-lived services require maintenance
- Platform was fragmenting
- New infrastructure updates were hard to apply

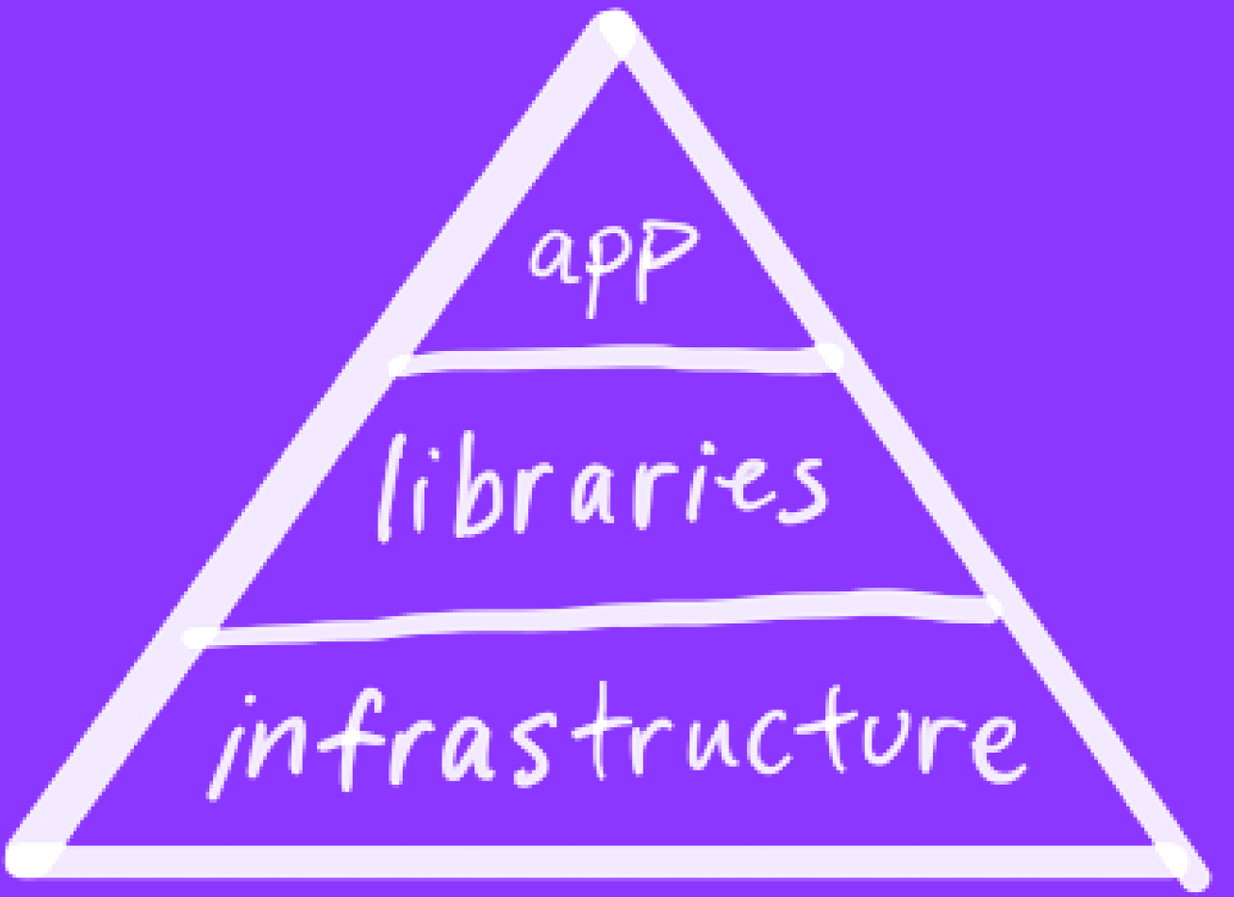
Technical leverage

But where to start?


app
libraries
infrastructure

empower

manage



Generation 3: @lyft/service ✨

Principles for Technical Leverage

Stand on the Shoulders of Giants

Simplify to Understand

Standardize and **Automate**

Stand on the Shoulders of Giants 

We chose **Next.js** as our platform of choice

Stand on the Shoulders of Giants 

We chose **Next.js** as our platform of choice

Solved: Build configurations, static site generation, AMP pages, code splitting, dynamic imports

Stand on the Shoulders of Giants 

We chose **Next.js** as our platform of choice

Solved: Build configurations, static site generation, AMP pages, code splitting, dynamic imports

 **Now:** We don't need to maintain our internal build system anymore

Simplify to Understand ✨

Paradigm shift: convention over configuration

Simplify to Understand ✨

Paradigm shift: convention over configuration

Next.js: Filesystem router, server-side getInitialProps and getServerSideProps handlers

Simplify to Understand ✨

Paradigm shift: convention over configuration

Next.js: Filesystem router, server-side getInitialProps and getServerSideProps handlers

🍕 **Now:** Lower cognitive load working in apps, higher developer productivity

Standardize and Automate 🤖

We built a **plugin** system that standardizes our library integrations

Standardize and Automate 🤖

We built a **plugin** system that standardizes our library integrations

We made **migrations** a first-class part of our new system

Standardize and Automate 🤖

We built a **plugin** system that standardizes our library integrations

We made **migrations** a first-class part of our new system

 **Now:** we have the tools to reuse code, keep the stack modern and prevent drift

Anatomy of a Plugin

- A set of hooks, bundled up in a library
- Plugin Hooks:
  - Webpack
  - Express Middleware
  - Next.js Configuration
  - Next.js Application
  - Next.js Document

```
// 1. Install the plugin in lyft.plugins.ts
import CookieAuthPlugin from "@lyft/service-plugin-cookie-auth";
const plugins = [
  new CookieAuthPlugin(),
  /* Other plugins */
];
```

```
// 2. Use it!
import { useCookieAuth } from "@lyft/service-plugin-cookie-auth";

// In React component
const Page: React.FC = () => {
  const { userId, isLoggedIn } = useCookieAuth();
  // That's it! You can now use as you see fit
  if (!isLoggedIn()) {
    return <p>Sorry, you must be logged in</p>;
  }
};
```

```ts
// CookieAuthPlugin: Express.js server hook
import cookieParser from "cookie-parser";
import { Application } from "express";

const cookieAuthServerHook = (app: Application) => {
  // Gives us req.cookies
  app.use(cookieParser);

  app.use(function parseUserId(req, res, next) {
    // Assume this decrypts data and returns a user ID from a session
    const { userId } = parseSessionCookies(req.cookies);

    // Store userId in response for later retrieval
     res.locals.userId = userId;
    next();
  });
};
```

```
// CookieAuthPlugin: Next.js App hook
function CookieAuthApp({ App: NextApp }) {
  return class extends App {
    static getInitialProps = async (appContext) => {
      const originalProps = await App.getInitialProps(appContext);
       const userId = appContext.ctx.res?.locals?.userId;

      return {  ...originalProps, userId };
    };

    render() {
      return (
          <CookieAuthContext.Provider value={this.props.userId}>
           {super.render()}
          </CookieAuthContext.Provider>
      );
    }
  };
}
```

```
// Bring it all together into the Plugin
export default class CookieAuthPlugin {
  apply = (service: ServicePluginHost) ⇒ {
    service.hooks.server.tap(this.name, cookieAuthServerHook);
    service.hooks.app.tap(this.name, (App) ⇒ CookieAuthApp({ App }));
  };
}
```

```
// And add a nice developer-facing convenience hook
const useCookieAuth = () ⇒ ({
  userId: React.useContext(CookieAuthContext),
  isLoggedIn: () ⇒ {
    const userId = React.useContext(CookieAuthContext);
    return !!userId;
  },
});
```

@lyft/service Plugin Ecosystem

- State management (Redux, MobX, XState)
- GraphQL
- Lyft Product Language, styled-components, Material UI
- authn/authz
- i18n
- RUM performance tracking
- Feature flagging and experimentation
- MirageJS
- Logging/metrics/bug reporting

Coming Soon

- Developer Support Tooling
- Embedded tools to help developers debug or ask for help

Flywheel effect

- Now users are contributing back to these plugins
- Over 40% of new plugins have been product-engineer contributions

Migrations - How we Automate

- Guardrails to prevent drift
- jscodeshift scripts

```
// Original
import { logger } from "@lyft/service-plugin-logging";
logger.info("test log");

// Upgraded
import { getLogger } from "@lyft/service-plugin-logging";
const logger = getLogger();
logger.info("test log");
```

Migration Versioning

- We use versioned migrations
- If you change an interface, you must ship a migration
- Store migration state per plugin in package.json

Release Management: One bold constraint

- The platform version and the plugin system are pinned to the same version 🎯
- Plugins are guaranteed to work with a specific version of the platform
- This means the entire platform moves together!

Organizational process

- Hands-on migration workshops
- Migration scripts take services most of the way from Gen 2 to Gen 3
- Relentless internal evangelism
- Technical program management + senior leadership visibility are key

Wins

- Higher developer happiness and productivity
- Quicker adoption of new service releases, preventing drift

Principles for Technical Leverage

Stand on the Shoulders of Giants

Simplify to Understand

Standardize and **Automate**