Rendering models that scale

Whatever your framework

Phil Hawksworth
Director of Developer Experience, Netlify

Finding the right approach for a project

Mo clients, mo problems

(Mo clients, mo architectures)

unsplash.com/photos/Bli_WDC2oU0

Web development is not Highlander

Finding the simplest solution

Finding the simplest solution

(implementation, cost, support, confidence, robustness)

Jamstack

Jamstack

> A way of thinking about how to build for the web. The UI is compiled, the frontend is decoupled, and data is pulled in as needed.

from

Generating responses to every request on demand

to

Generating responses to requests in advance

to

Generating responses to requests in advance

(You know... like we did in the 90s)

BUZZWORD BINGO

Pre-rendering

WHEN?

Build time vs Request time

Front-end code is no longer limited to being a product of a **back-end** system

WHERE?

Application server

VS

Serverless functions

BUZZWORD BINGO

Front-end code is no longer limited to being a product of a **back-end** system

...I love these discussions!

@PhilHawksworth

Director of Developer Experience, Netlify

findthat.at/**jamstack/book**



REMOTELY INTERESTING




findthat.at/interesting

Let’s talk

1. Benefits and limits
2. Breaking the ceiling
3. Removing the ceiling

1

Benefits and limits

Security

How susceptible our infrastructure is to attack

Scale

How well we can handle high volumes of traffic

Speed

How quickly we can deliver content to the users

Sperience

Workflow
Developer experience

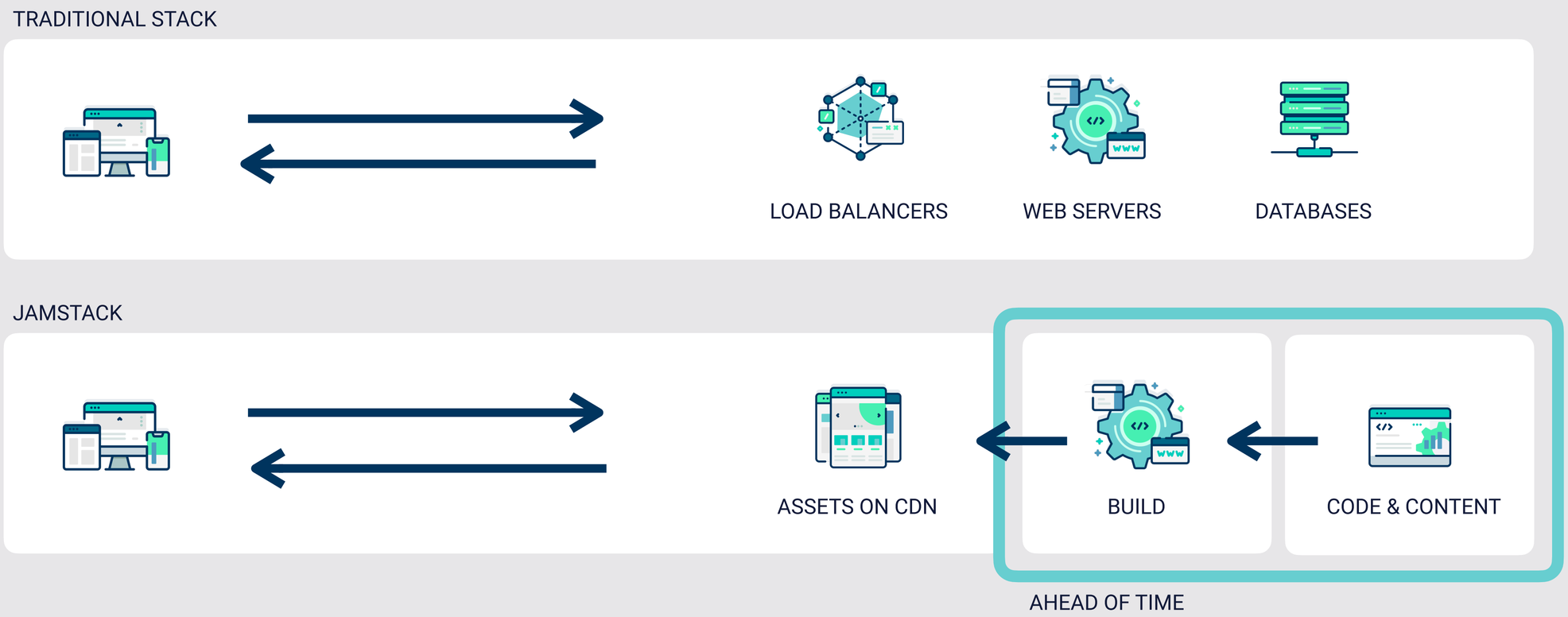
How efficiently and confidently we can build, release, and maintain sites

Benefit

Simplicity

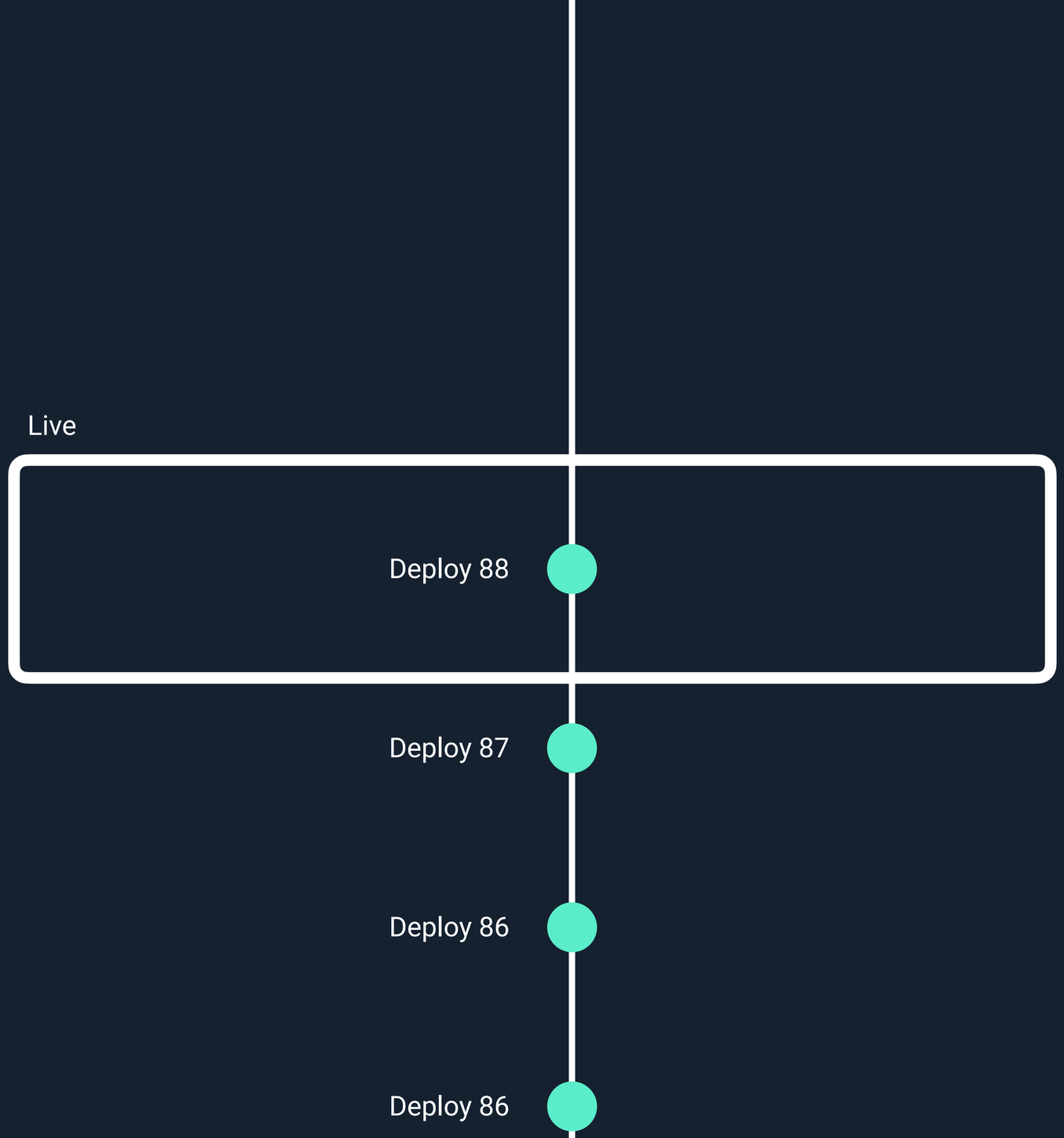
TRADITIONAL STACK

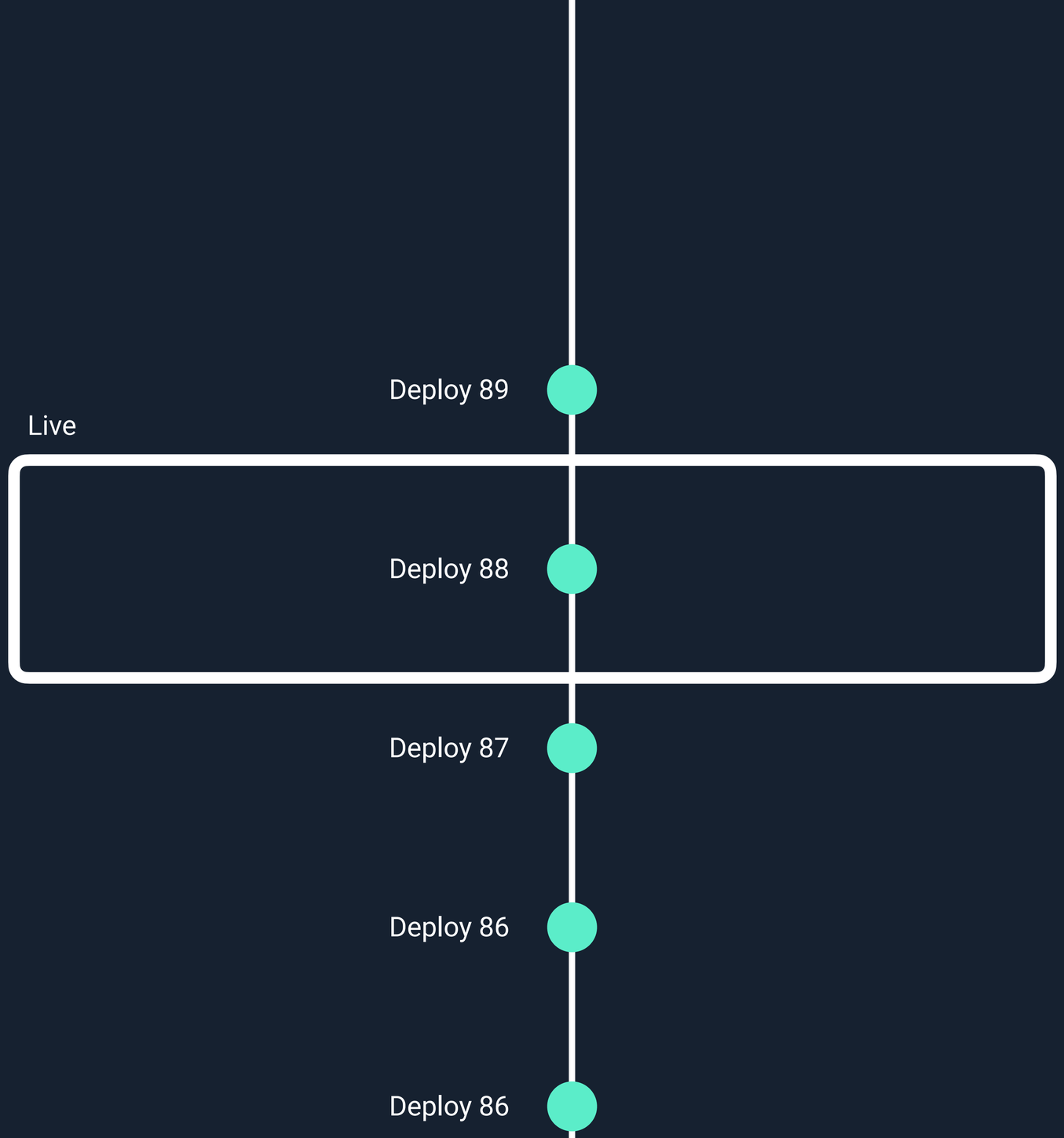
JAMSTACK

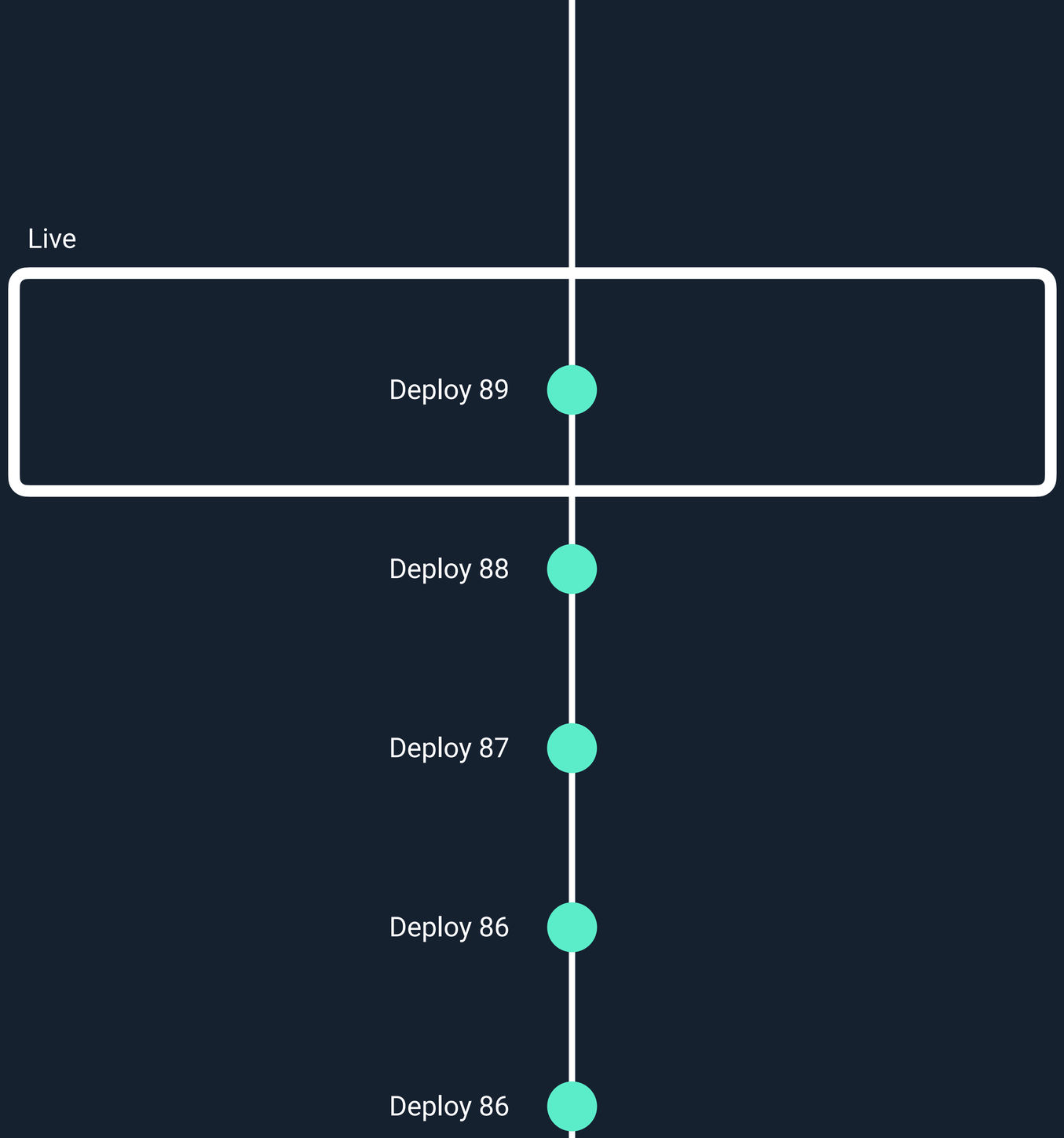

TRADITIONAL STACK
LOAD BALANCERS
WEB SERVERS
DATABASES
JAMSTACK
ASSETS ON CDN
BUILD
CODE & CONTENT
AHEAD OF TIME

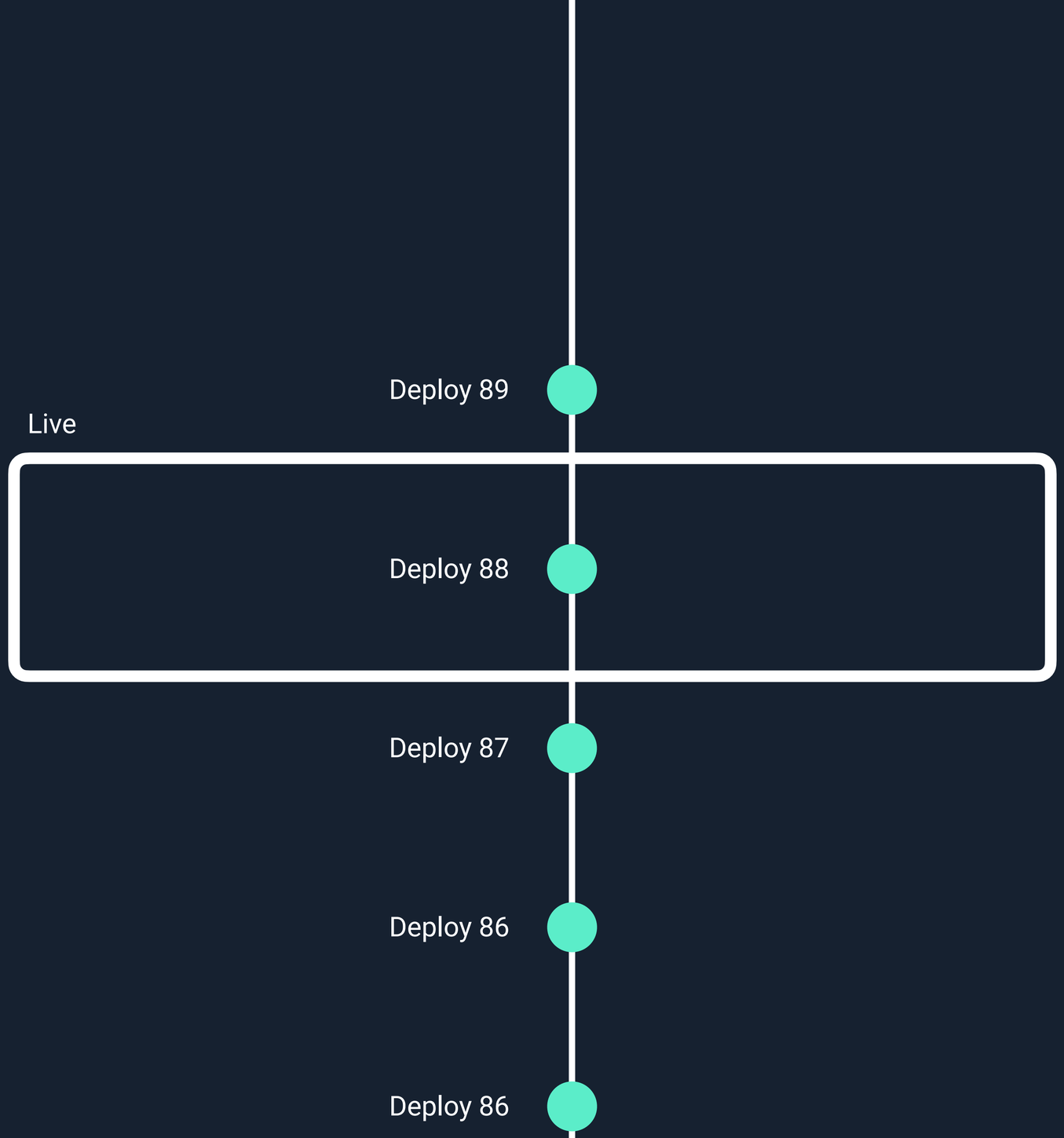
Benefit

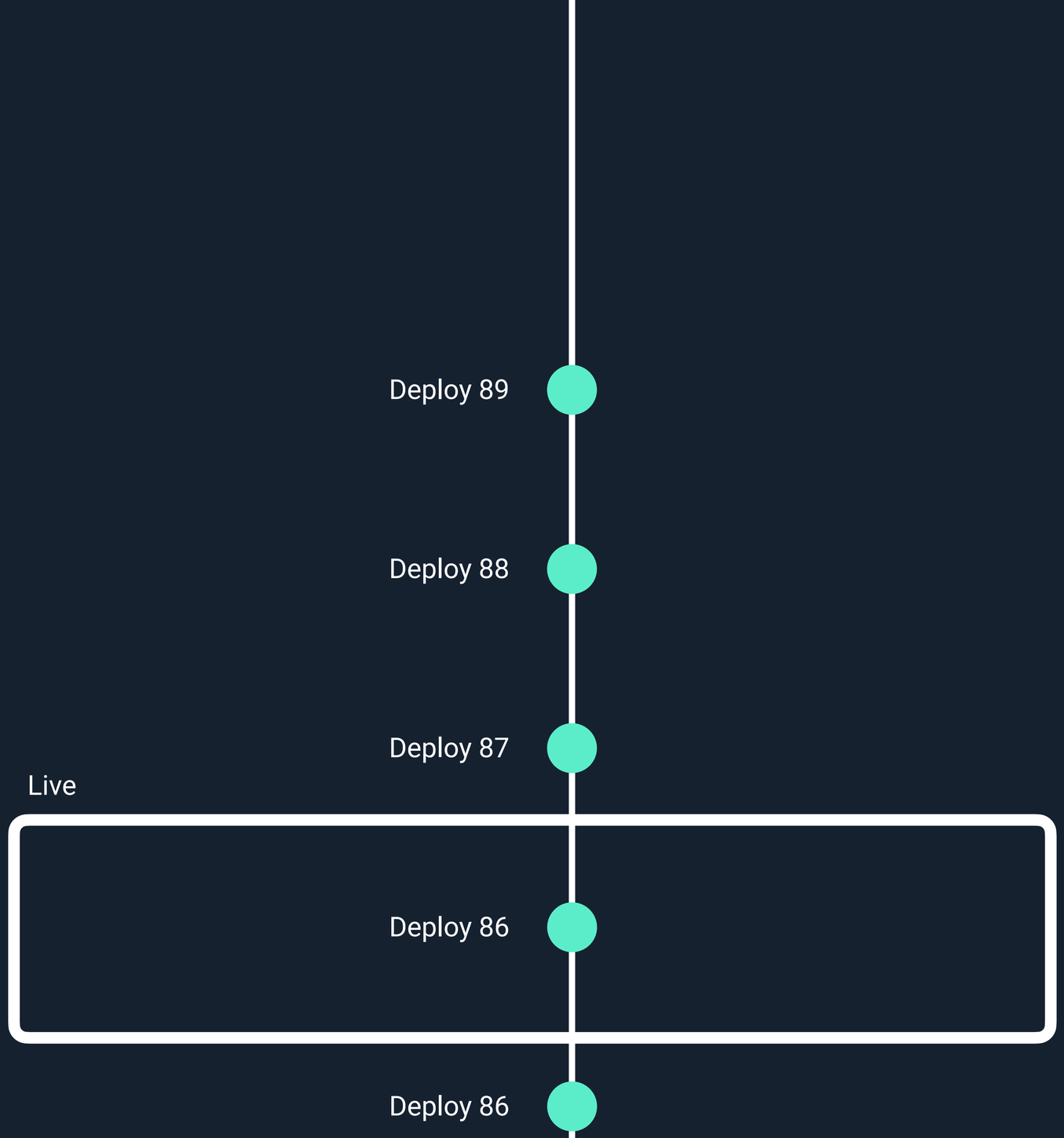
Immutable, Atomic deploys


Live
Deploy 88
Deploy 87
Deploy 86
Deploy 86


Deploy 89
Live
Deploy 88
Deploy 87
Deploy 86
Deploy 86


Live
Deploy 89
Deploy 88
Deploy 87
Deploy 86
Deploy 86


Deploy 89
Live
Deploy 88
Deploy 87
Deploy 86
Deploy 86


Deploy 89
Deploy 88
Deploy 87
Live
Deploy 86
Deploy 86

Deploys are immutable and atomic


JAMSTACK
ASSETS ON CDN
BUILD
CODE & CONTENT
AHEAD OF TIME

Challenge

Very large sites

Challenge

User generated content

We reach a ceiling

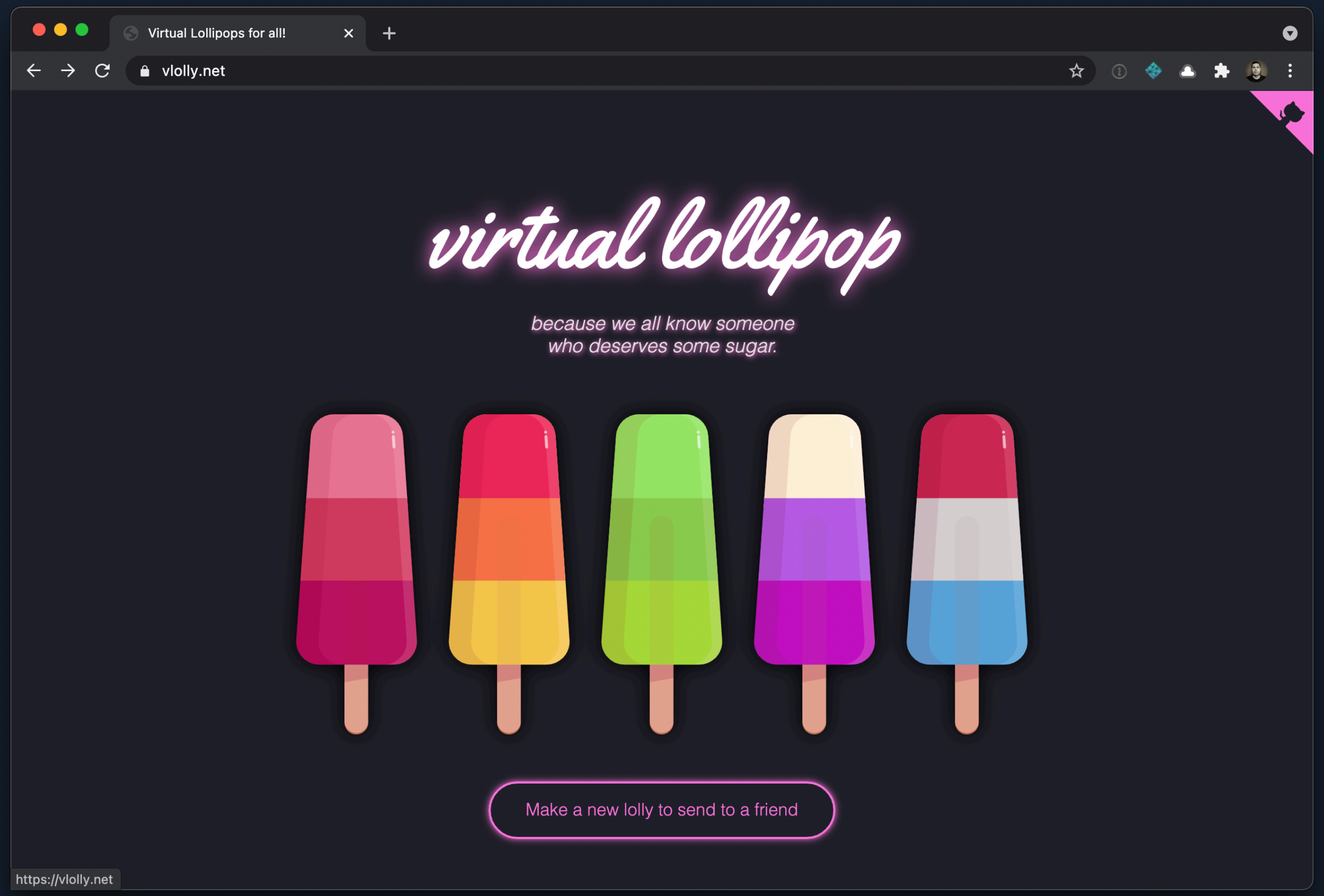
unsplash.com/photos/unIuWzRFaSM

2

Breaking the ceiling

vlolly.net

findthat.at/lollytricks

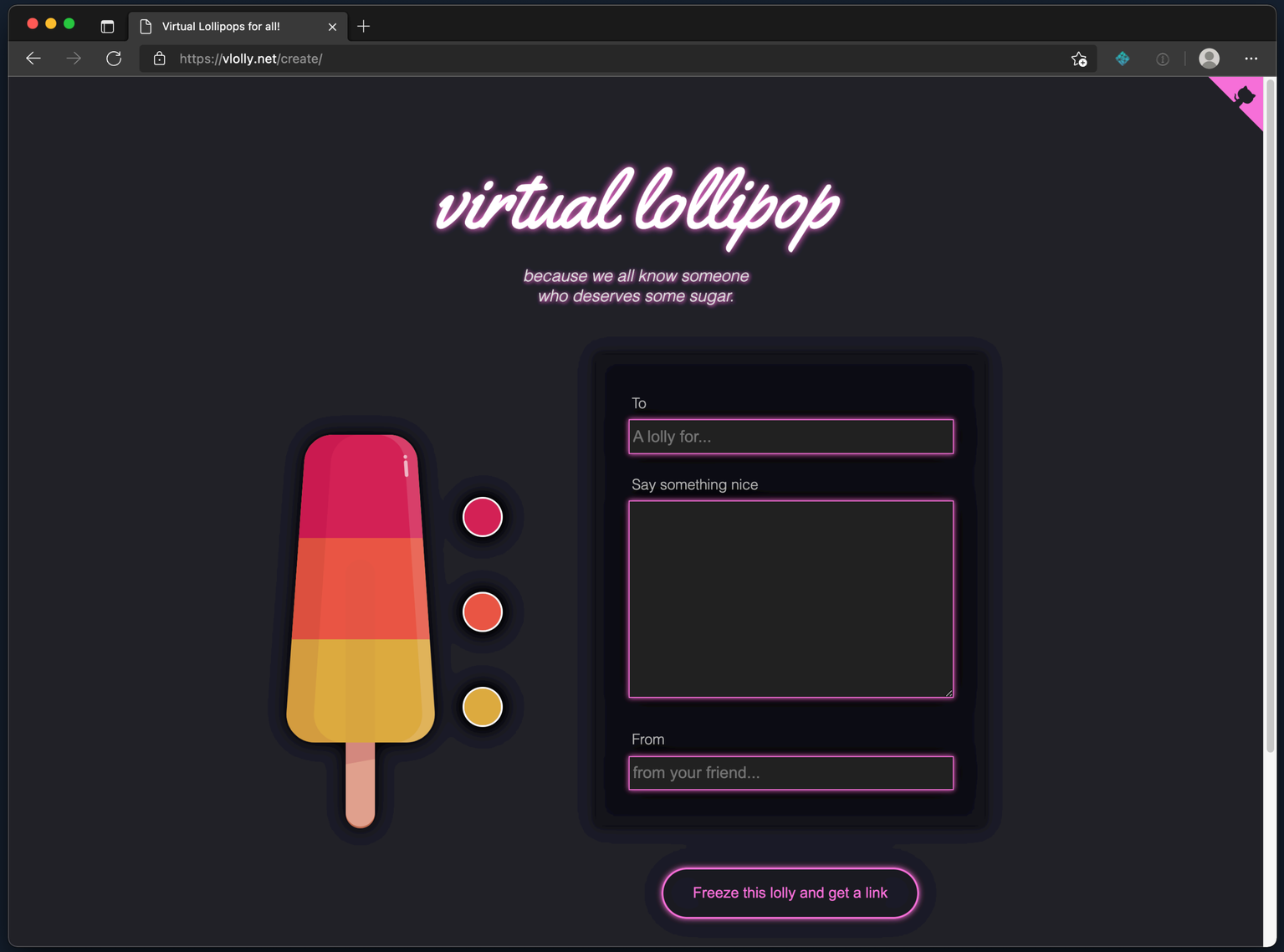


Thousands of unique pages and user-generated content

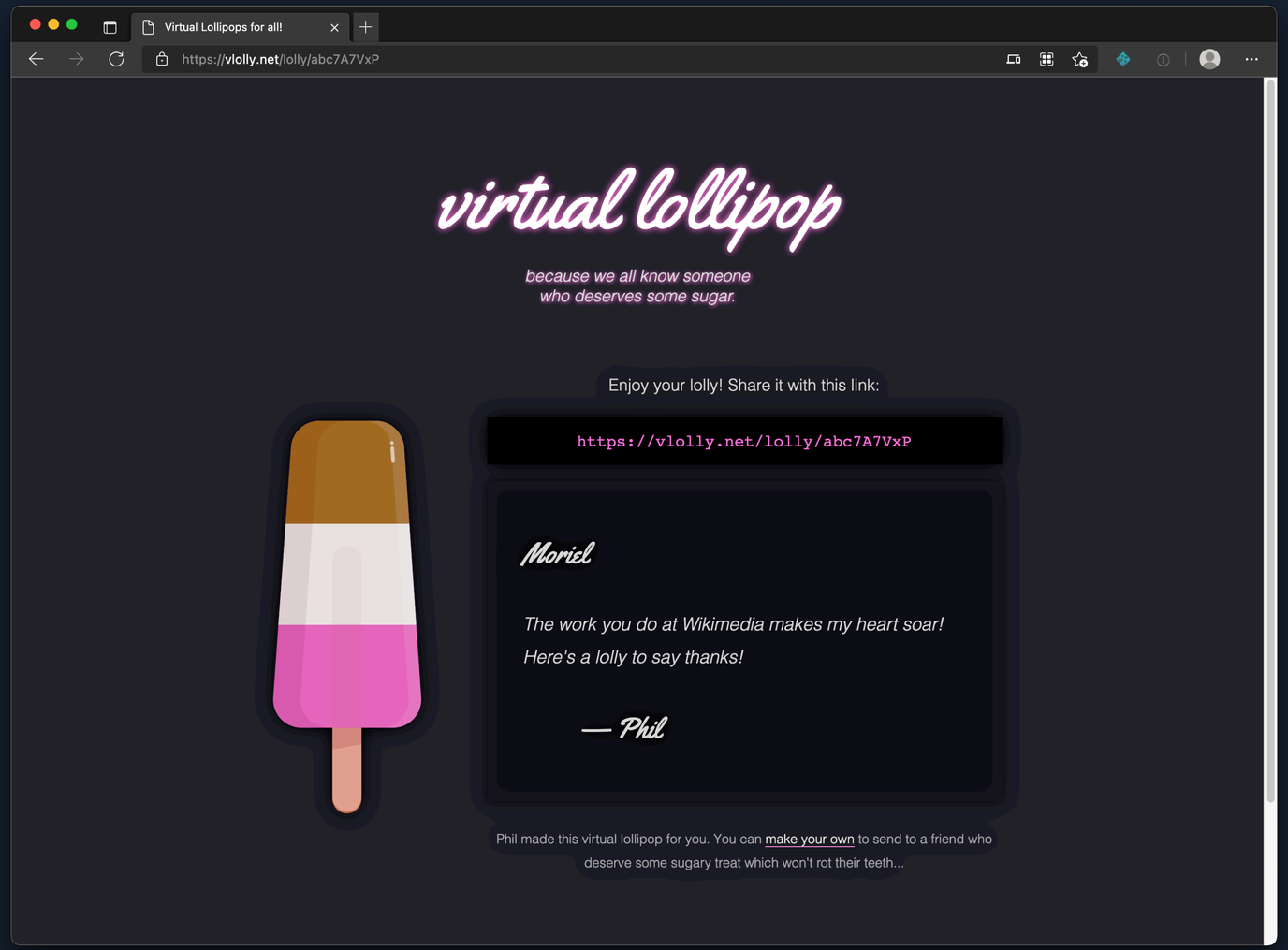
New content rendered on-demand as a fallback

New content rendered on-demand as a fallback

New content rendered on-demand as a fallback


Virtual Lollipops for all!
https://vlolly.net/create/
virtual lollipop
because we all know someone
who deserves some sugar.
To
A lolly for...
Say something nice
From
from your friend...
Freeze this lolly and get a link

vlolly.net/lolly/abc7A7VxP



New content

Rebuild and fill the gaps

404

Render page on demand

functions/showLolly.js

```
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');

// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});

// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];

  // find the lolly data in the DB
  client.query(
    q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
  ).then((response) => {

    // if found, return a view
    return {
      statusCode: 200,
      headers: {
          "Content-Type": "text/html",
      },
      body: pageTemplate(response.data)
    }

  }).catch((error) => {
    // not found or an error, send to the generic error page
  });

}

// export the handler function
exports.handler = handler;
```

```
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');
```

functions/showLolly.js

```
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');

// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});

// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];

  // find the lolly data in the DB
  client.query(
    q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
  ).then((response) => {

    // if found, return a view
    return {
      statusCode: 200,
      headers: {
          "Content-Type": "text/html",
      },
      body: pageTemplate(response.data)
    }

  }).catch((error) => {
    // not found or an error, send to the generic error page
  });

}

// export the handler function
exports.handler = handler;
```


```
// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});
```

functions/showLolly.js

```
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');

// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});

// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];

  // find the lolly data in the DB
  client.query(
    q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
  ).then((response) => {

    // if found, return a view
    return {
      statusCode: 200,
      headers: {
          "Content-Type": "text/html",
      },
      body: pageTemplate(response.data)
    }

  }).catch((error) => {
    // not found or an error, send to the generic error page
  });

}

// export the handler function
exports.handler = handler;
```


```
// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];
```

functions/showLolly.js

```
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');

// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});

// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];

  // find the lolly data in the DB
  client.query(
    q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
  ).then((response) => {

    // if found, return a view
    return {
      statusCode: 200,
      headers: {
          "Content-Type": "text/html",
      },
      body: pageTemplate(response.data)
    }

  }).catch((error) => {
    // not found or an error, send to the generic error page
  });

}

// export the handler function
exports.handler = handler;
```

```
// find the lolly data in the DB
client.query(
  q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
).then((response) => {
```

functions/showLolly.js

```javascript
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');

// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});

// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];

  // find the lolly data in the DB
  client.query(
    q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
  ).then((response) => {

    // if found, return a view
    return {
      statusCode: 200,
      headers: {
          "Content-Type": "text/html",
      },
      body: pageTemplate(response.data)
    }

  }).catch((error) => {
    // not found or an error, send to the generic error page
  });

}

// export the handler function
exports.handler = handler;
```

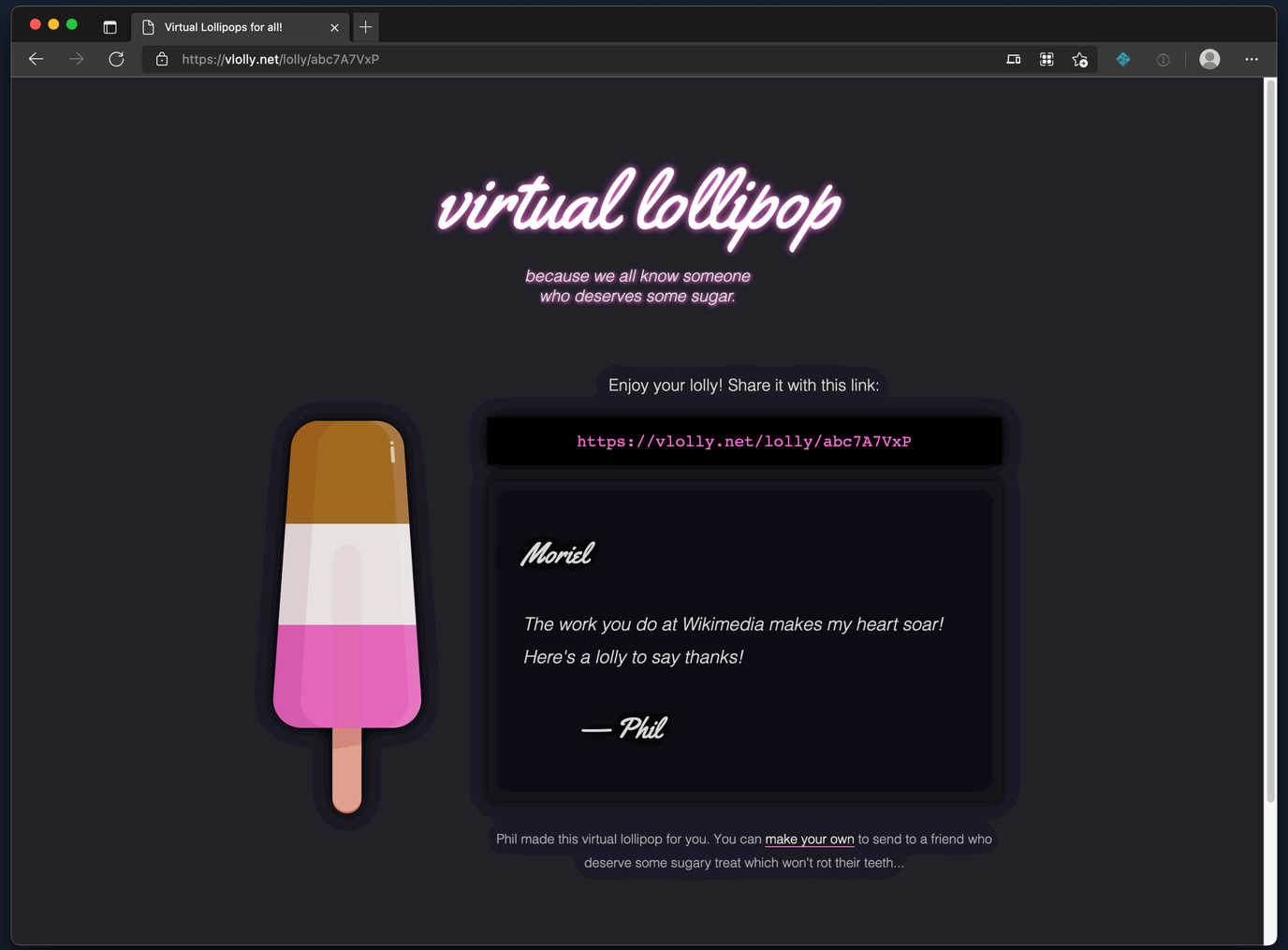
```javascript
// if found, return a view
return {
  statusCode: 200,
  headers: {
    "Content-Type": "text/html",
  },
  body: pageTemplate(response.data)
}
```

It’s JavaScript

Add this to any framework or site

```toml
# netlify.toml
# lolly page requests will be rendered and persisted on demand
[[redirects]]
  from = "/lolly/*"
  to = "/.netlify/functions/showLolly"
  status = 200
```

vlolly.net/lolly/abc7A7VxP



New content

Rebuild and fill the gaps

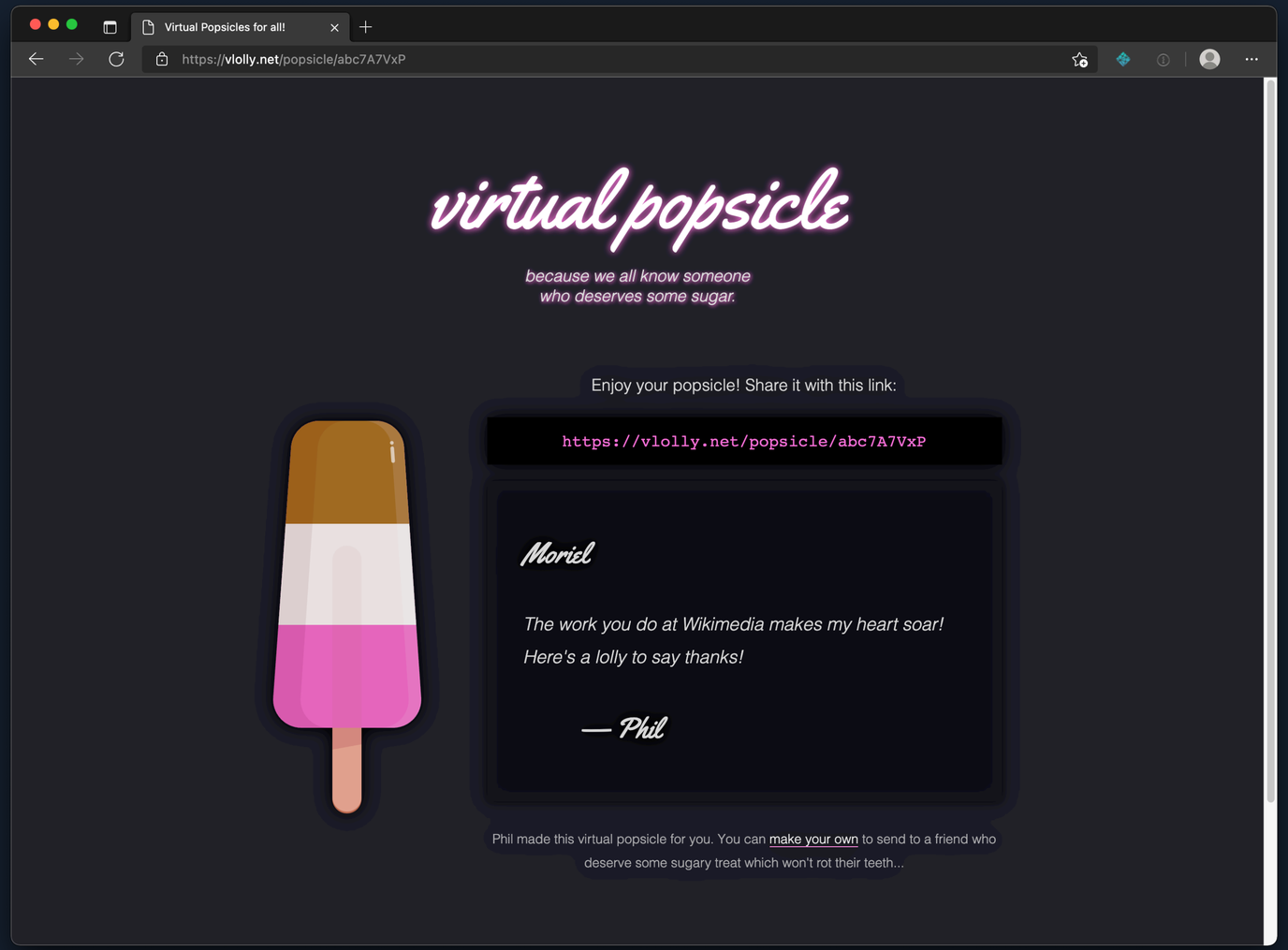
404

Render page on demand

Lots of pages

Cache the builds

vlolly.net/popsicle/abc7A7VxP



(Making-fun-of-my-english-accent-driven development)

It’s JavaScript

Add this to any framework or site

But...

More pages means longer builds

We could have better cache characteristics

Complexity starting to muddy the mental model

I'm inventing things rather than using a formal pattern

But behold, there are now

Numerous rendering approaches

ISR DSG DPR

3

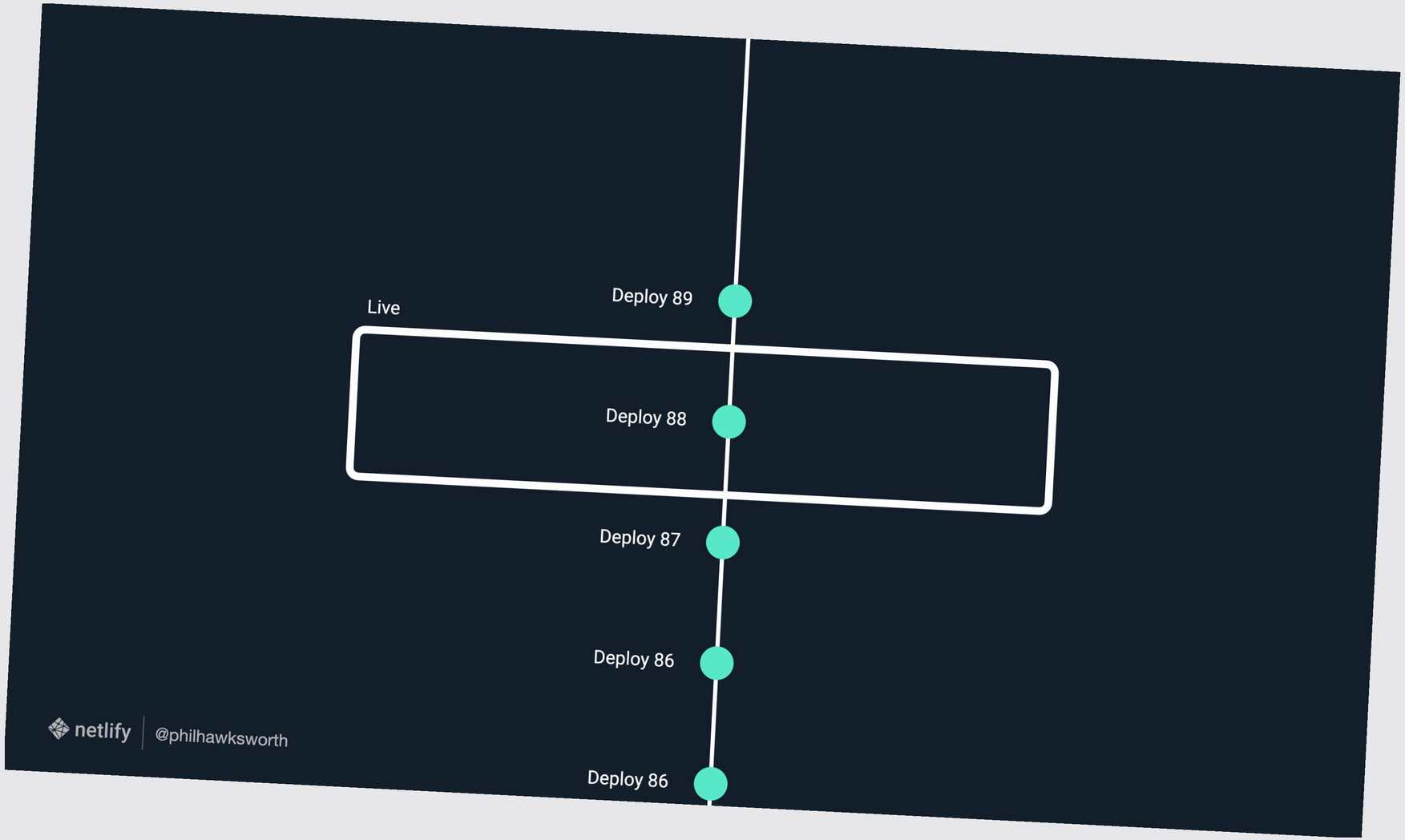
Removing the ceiling

ISR DSG DPR

Without undoing the benefits of Jamstack

Deploys are immutable and atomic

netlify | @philhawksworth


Live
Deploy 89
Deploy 88
Deploy 87
Deploy 86
Deploy 86
netlify
@philhawksworth

Without migrating to another framework

DPR

Distributed Persistent Rendering

DPR

Distributed Persistent Rendering

On-demand Builders | Netlify D

https://docs.netlify.com/configure-builds/on-demand-builders

netlify docs

Search our docs by topic...

Log in Sign up →

- Home
- Configure builds
- Site deploys
- Monitor sites
- Domains & HTTPS
- Static routing

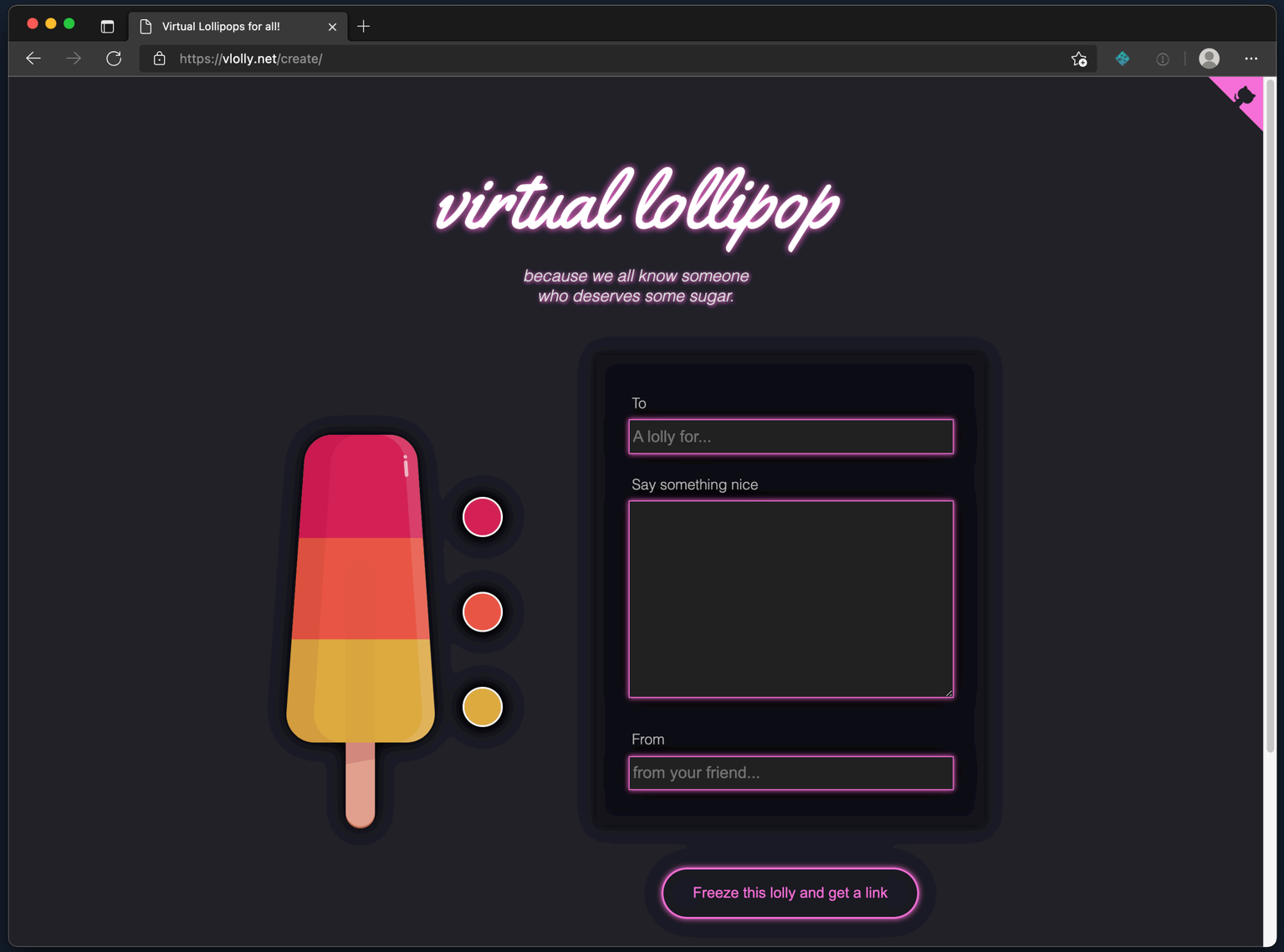
On-demand Builders

This feature is in `BETA`.

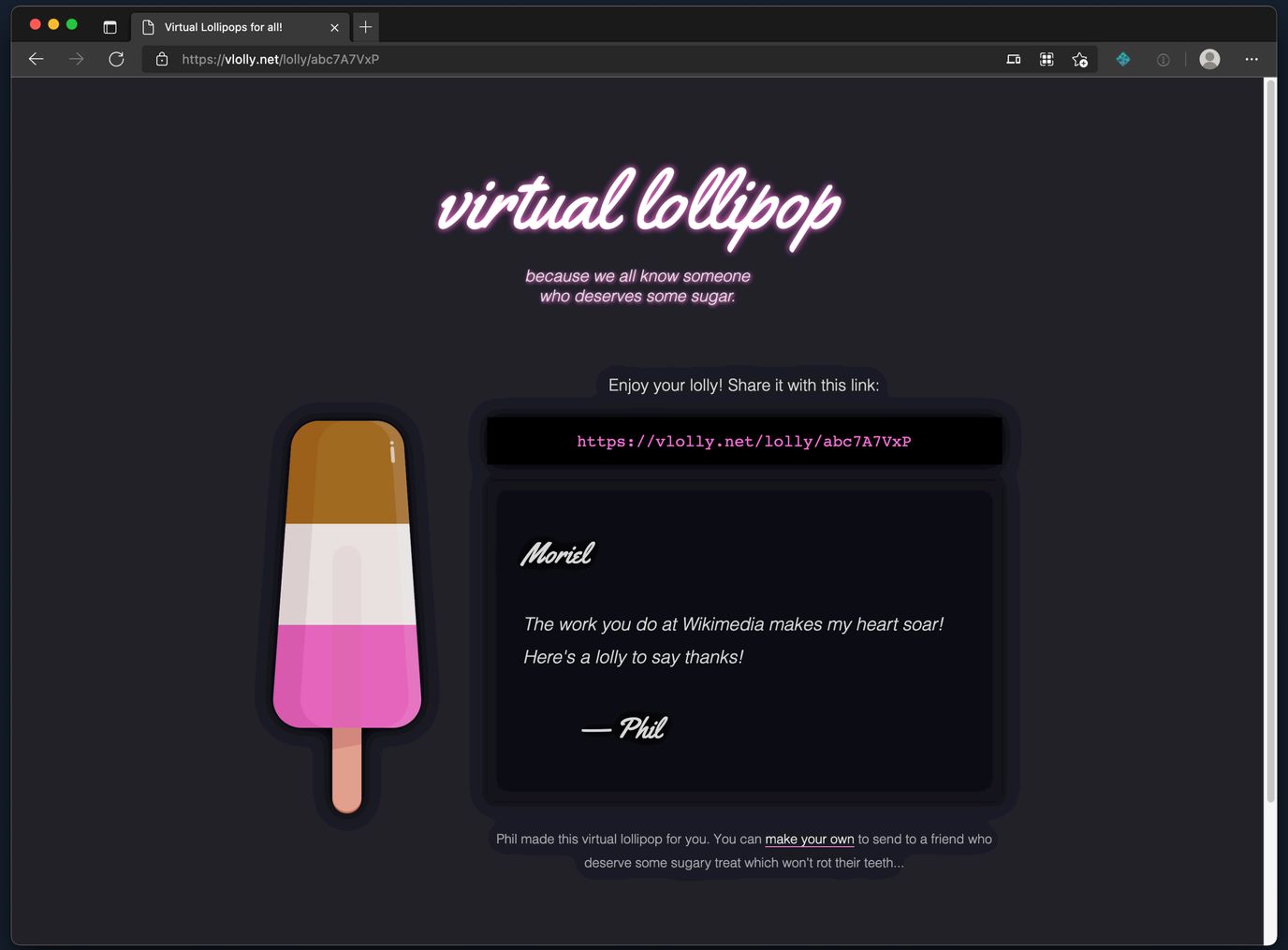
On-demand Builders are serverless functions used to generate web content as needed that's automatically cached on Netlify's Edge CDN. They enable you to build pages for your site when a user visits them for the first time and then cache them at the edge for subsequent visits.

ON THIS PAGE

- Create On-demand Builders
- Use On-demand Builders
  - Customize paths with redirects
- Frequently asked questions


Virtual Lollipops for all!
https://vlolly.net/create/
virtual lollipop
because we all know someone
who deserves some sugar.
To
A lolly for...
Say something nice
From
from your friend...
Freeze this lolly and get a link

vlolly.net/lolly/abc7A7VxP



functions/showLolly.js

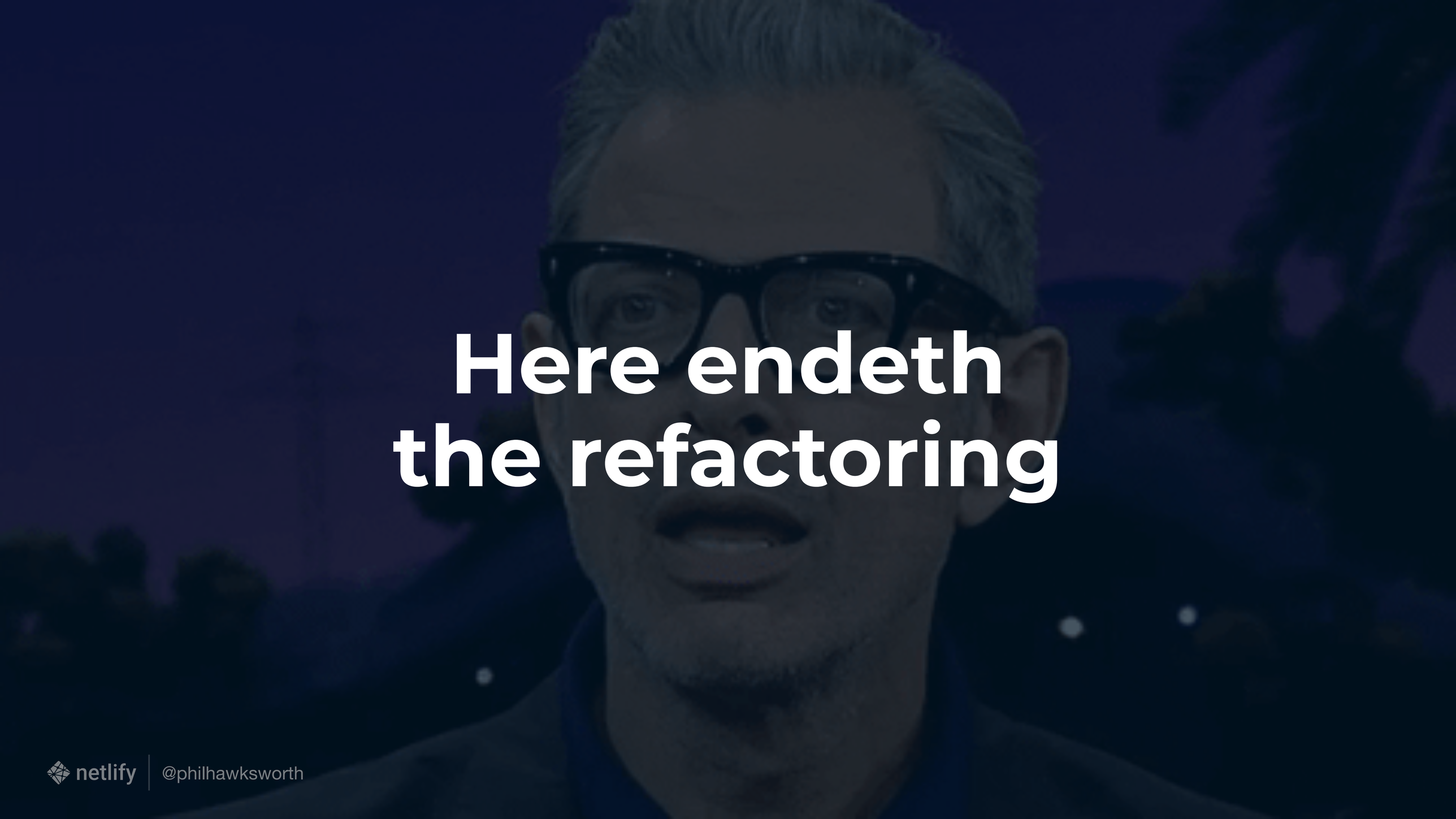
```
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');
const { builder } = require('@netlify/functions');

// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});

// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];

  // find the lolly data in the DB
  client.query(
    q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
  ).then((response) => {

    // if found, return a view
    return {
      statusCode: 200,
      headers: {
          "Content-Type": "text/html",
      },
      body: pageTemplate(response.data)
    }

  }).catch((error) => {
    // not found or an error, send to the generic error page
  });

}

// export the handler function
exports.handler = builder(handler);
```

```
const { builder } = require('@netlify/functions');
```

functions/showLolly.js

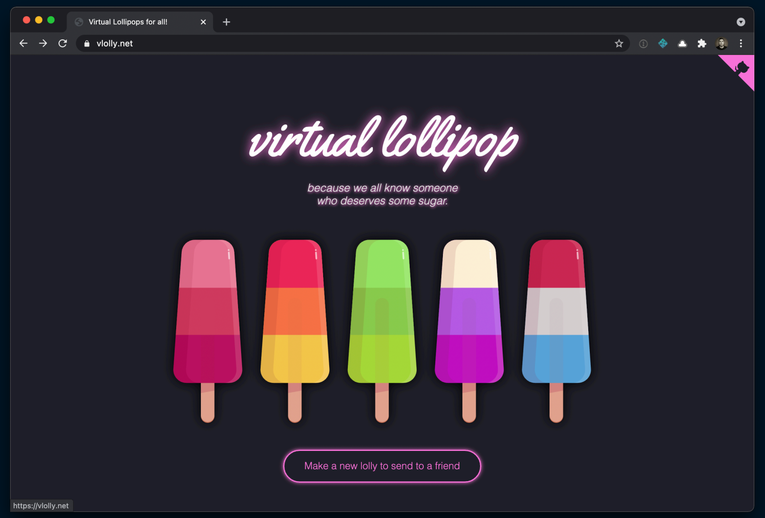
```
const faunadb = require('faunadb');
const pageTemplate = require('./lollyTemplate.js');
const { builder } = require('@netlify/functions');

// setup and auth the Fauna DB client
const q = faunadb.query;
const client = new faunadb.Client({
  secret: process.env.FAUNADB_SERVER_SECRET
});

// Create a handler function to export
const handler = async(event) => {

  // get the lolly ID from the request
  let lollyId = event.path.split("lolly/")[1];

  // find the lolly data in the DB
  client.query(
    q.Get(q.Match(q.Index("lolly_by_path"), lollyId))
  ).then((response) => {

    // if found, return a view
    return {
      statusCode: 200,
      headers: {
          "Content-Type": "text/html",
      },
      body: pageTemplate(response.data)
    }

  }).catch((error) => {
    // not found or an error, send to the generic error page
  });

}

// export the handler function
exports.handler = builder(handler);
```

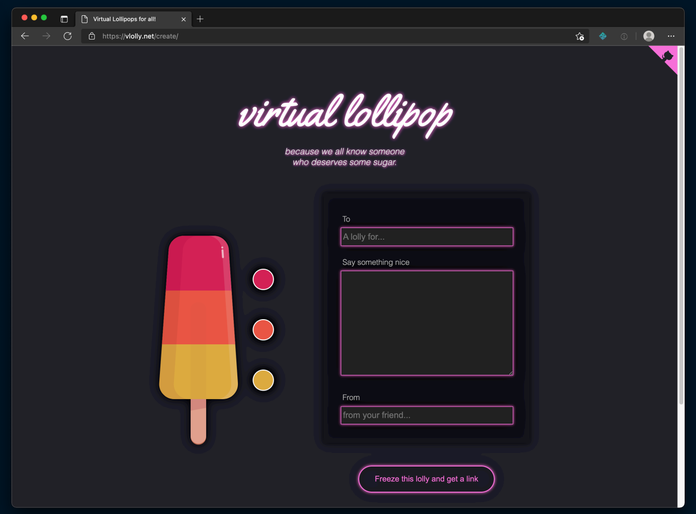
```
// export the handler function
exports.handler = builder(handler);
```



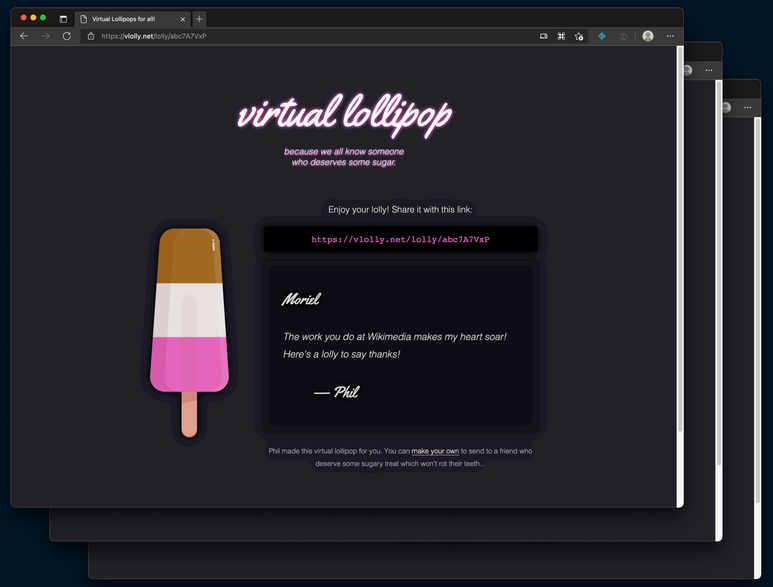
Here endeth the refactoring

Generated whenever we deploy





Added to the deploy when first requested



New content

Render into cache on demand

404

Nope. Page delivered from deploy

Lots of pages

Fewer pages built in deploys

tl;dr:

I deleted some stuff and then added 2 lines

This additive approach uses on-demand builders

It can be augment a site built with any framework

It doesn’t compromise the core benefits of Jamstack

Next?

Links

Run Gatsby 4 with DSG and SSR on Netlify

https://www.netlify.com/blog/2021/09/16/run-gatsby-4-with-dsg-and-ssr-on-netlify-today/

Keeping things fresh with stale-while-revalidate

https://web.dev/stale-while-revalidate/

Use Next.js 12 on Netlify

https://www.netlify.com/blog/2021/10/27/use-next.js-12-on-netlify/

Eleventy Serverless

https://www.11ty.dev/docs/plugins/serverless/

Distributed Persistent Rendering: A new Jamstack approach for faster builds

https://findthat.at/dpr

Distributed Persistent Rendering RFC

https://findthat.at/dpr/rfc

Thank you

@philhawksworth