Sitecore Experience Edge

Running Sitecore “high available at scale” made easy

delaware / 23.12.21

delaware

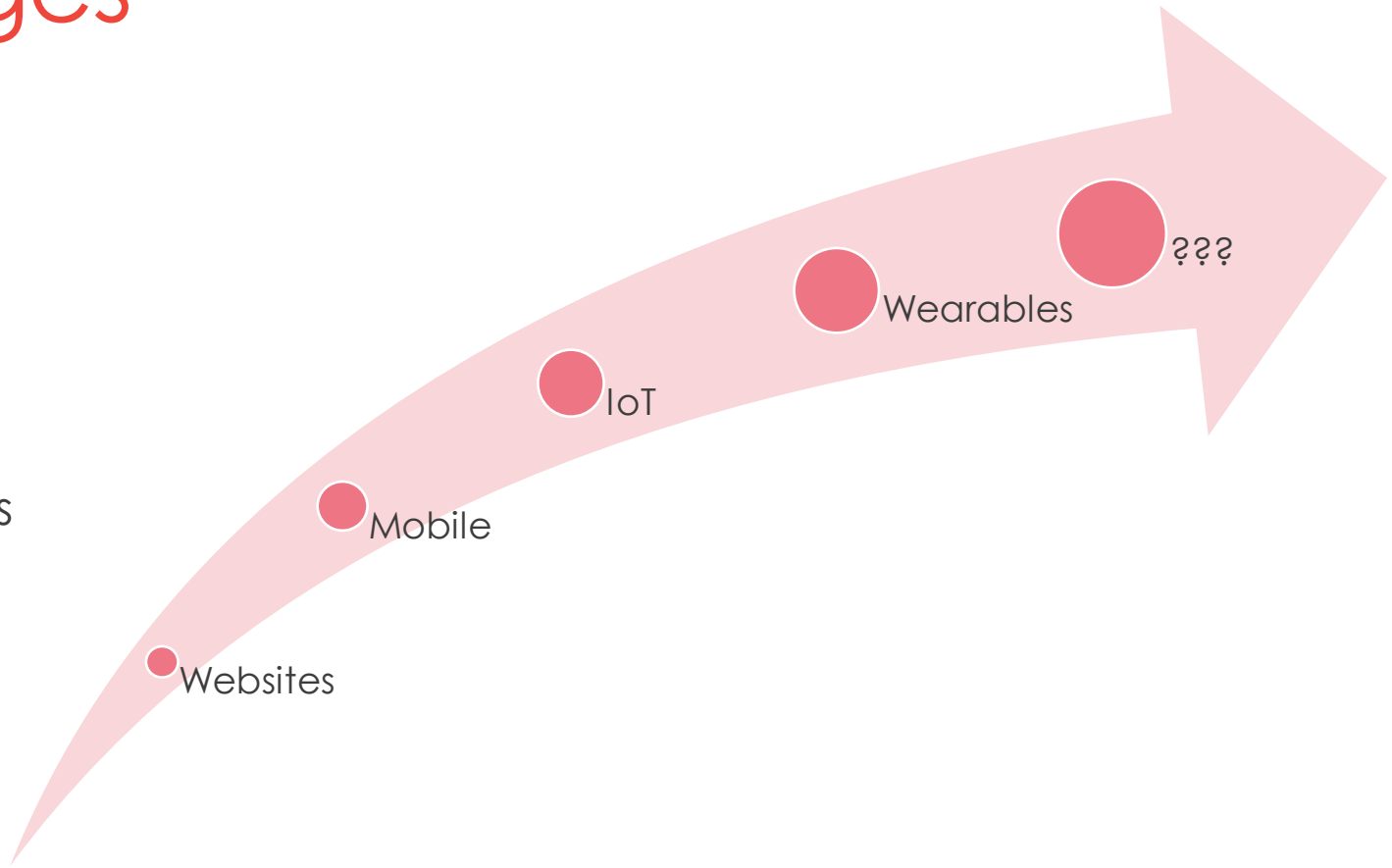
Outline

1. **Scaling Sitecore**
   From a traditional .NET MVC perspective
2. **Sitecore Experience Edge**
   Global reach
3. **Demo**
   delaware POC showcase
4. **Questions**
   ???

Today's challenges

"The evolution of the internet"

- More channels more visitors
  - Global reach
- High expectations
  - Seamless and fast experiences
- Business under pressure
  - Fast changing industry trends
  - E.g., covid19



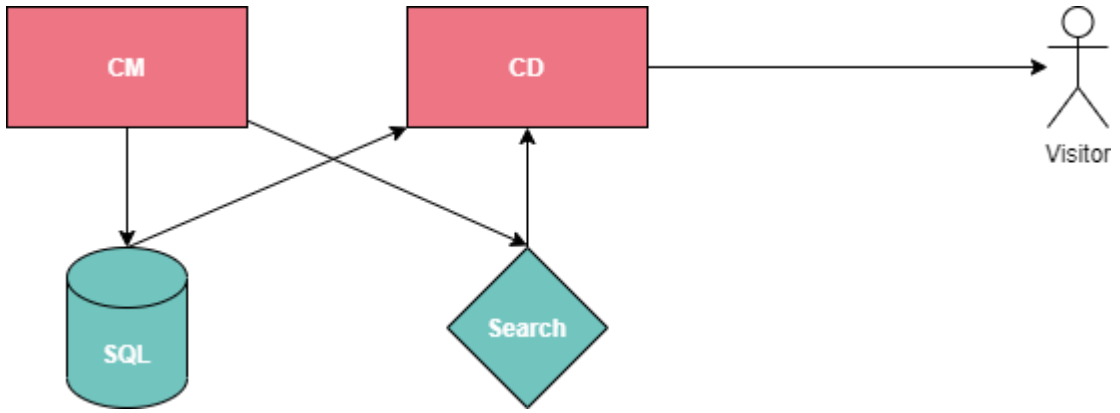
DXP architecture that can scale and serve globally. That can adapt to changing needs.

- Infrastructure, development, skills, …

Traditional Sitecore on prem

Limited infrastructure

- "Fixed" hardware (specs)
- "Fixed" topology (in a single region)



Flexible application

- Multi-site
- Multi-device

Sitecore solved complex business needs at application level

Infrastructure was limited / simple (and may still be valid for your current use case)

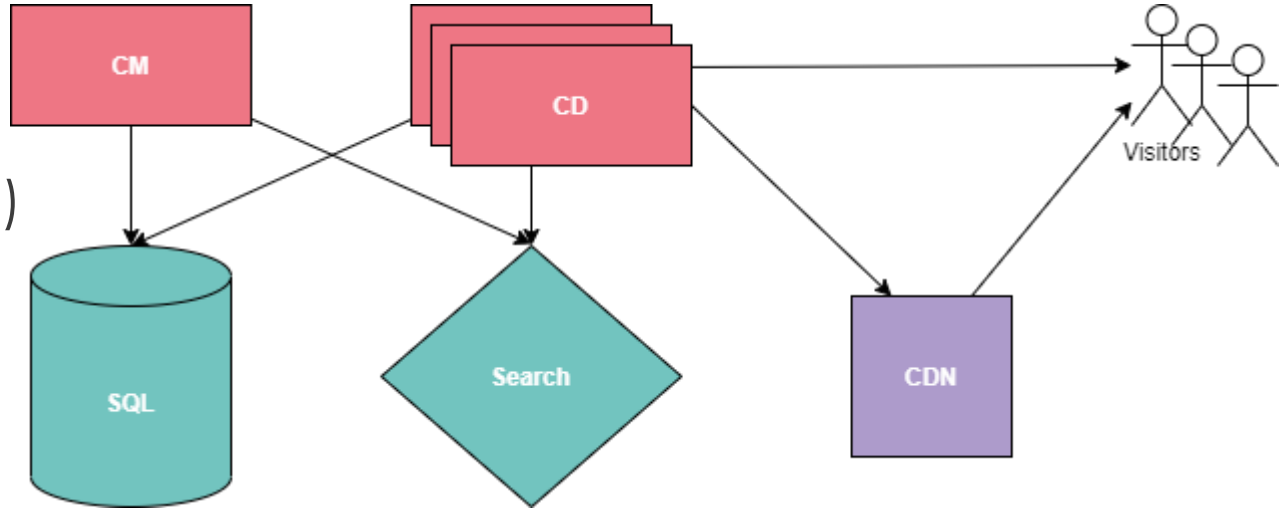
Traditional Sitecore in the cloud

Less limited infrastructure

- Scalable hardware
- "Less fixed" topology (still single-region)

Flexible application

- Multiple CD
- CDN



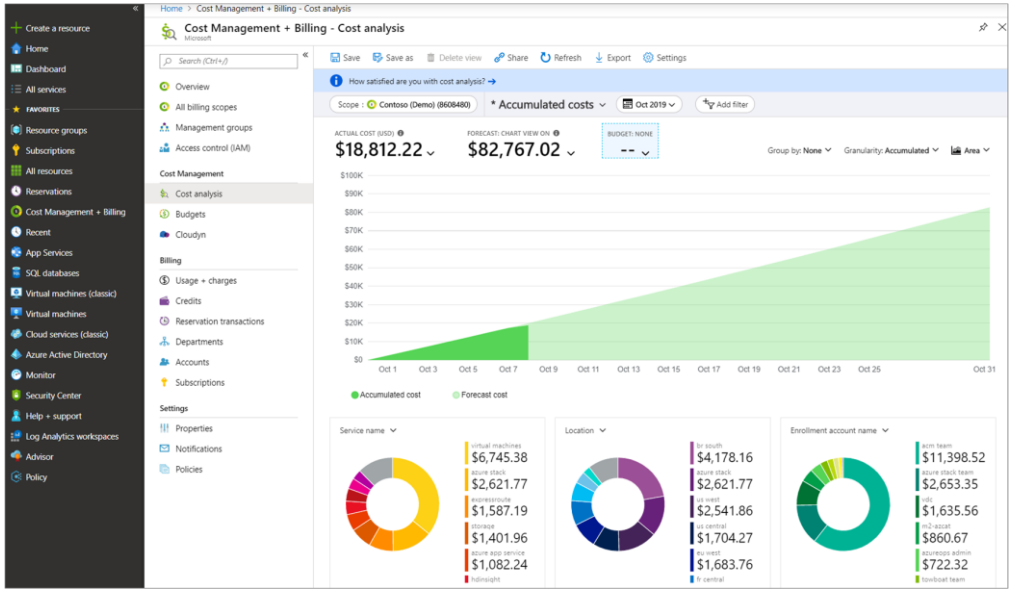
Cloud infrastructure comes with vertical scaling out of the box

Sitecore application supports horizontal scaling

Sitecore and global reach

Traditional Sitecore in the cloud makes it easy to scale at regional level

- Keep adding more hardware resources
- Is this the most cost-efficient way?



Regional presence might not be what business wants

- Does regional make sense from ROI perspective?
  - Marginal cost for regional reach increases
- Is your business (still) regional?
  - Growth, industry changes (e.g., covid), …

Global presence might be expected

- Customers, Google (Core Web Vitals), …

How to get global reach with Sitecore in a cost-effective way???

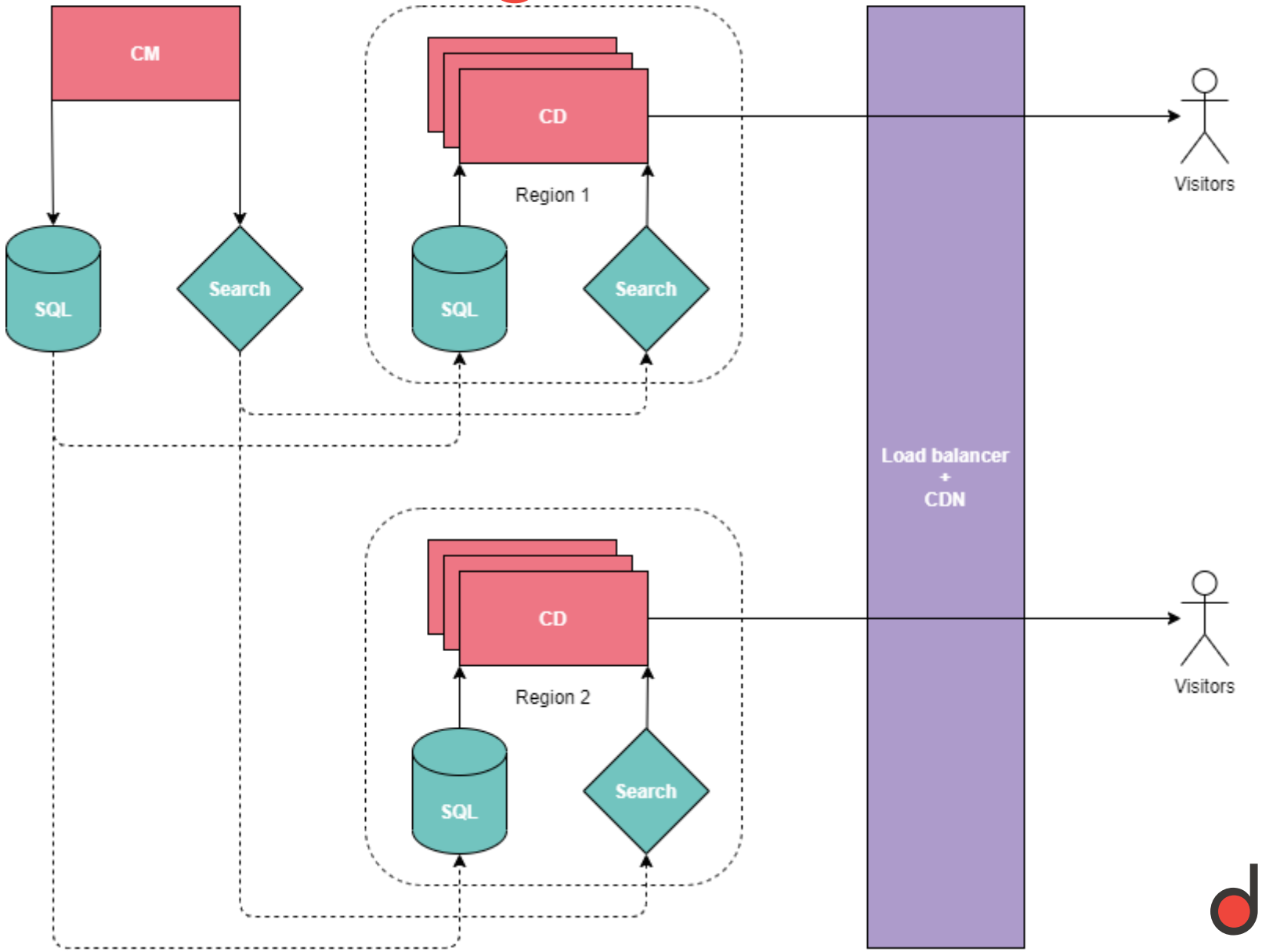
Traditional Sitecore cloud w/ global reach

Replicated infrastructure

- Setup per region
- Load balancer routes visitors to nearest region

Application supports

- Multiple regions
  - Publish to multiple DBs
  - Index multiple indexes
- Infrastructure replication
  - Azure DB geo-replication



Modern Sitecore: headless

Moves rendering from Sitecore CD to separate rendering host(s)

- Sitecore content is exposed through an API (true Content Delivery)
- Rendering host renders HTML from exposed content
- Rendering host can be anything: server (site), app (client), device (client), …

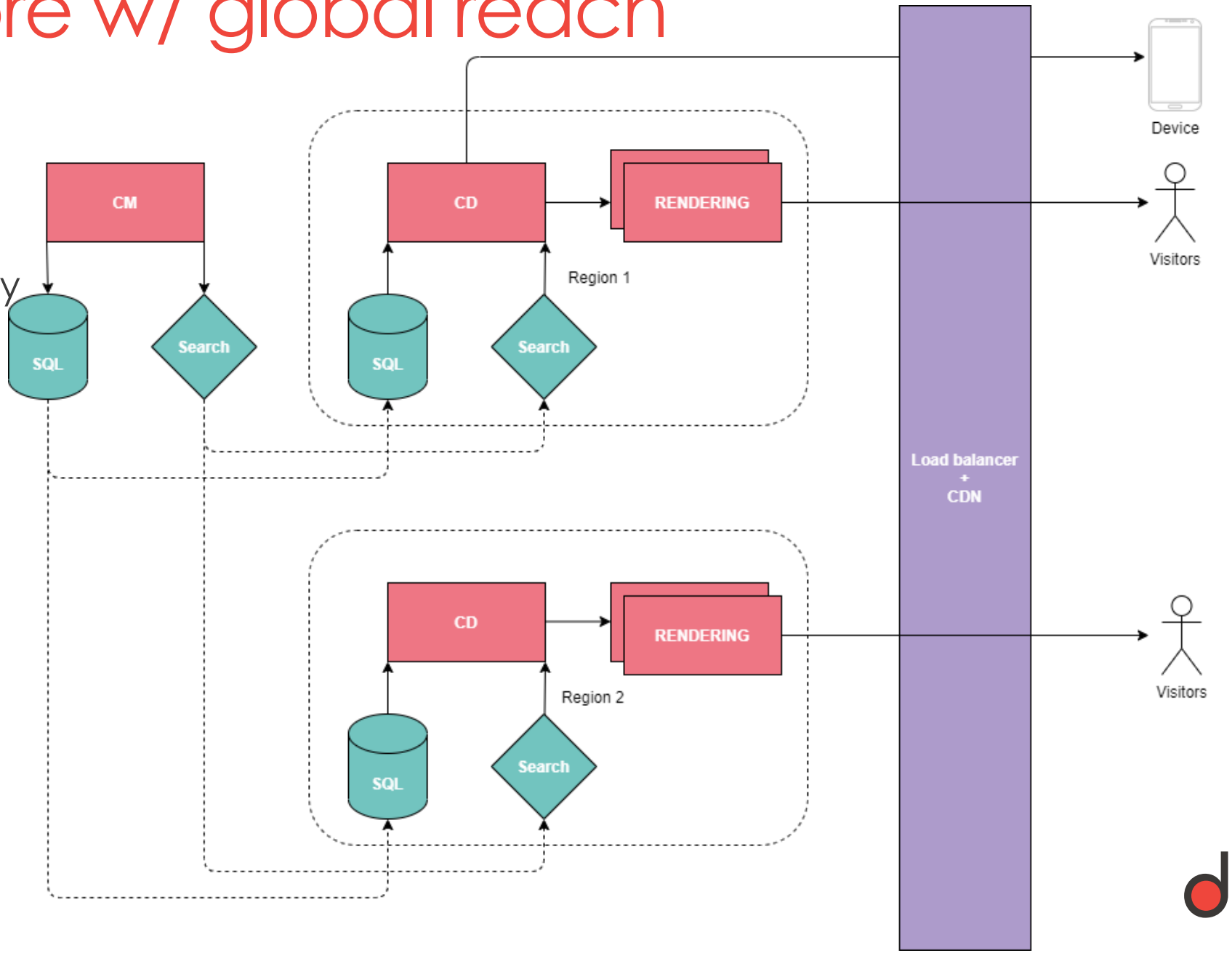
This improves

- Scaling on regional level: first scale rendering host, then scale Sitecore CD
  - Rendering host is probably more lightweight than your Sitecore CD
- Development agility: rendering expertise instead of Sitecore expertise
  - Faster delivery / release (debatable)
  - Modern frameworks and delivery infrastructure

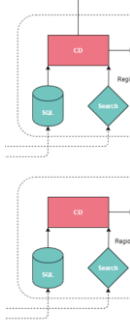
Modern Sitecore w/ global reach

This does not solve

- True global reach
- Infrastructure complexity
- Infrastructure cost?



Sitecore Experience Edge



"High available, scalable, SaaS delivery solution"

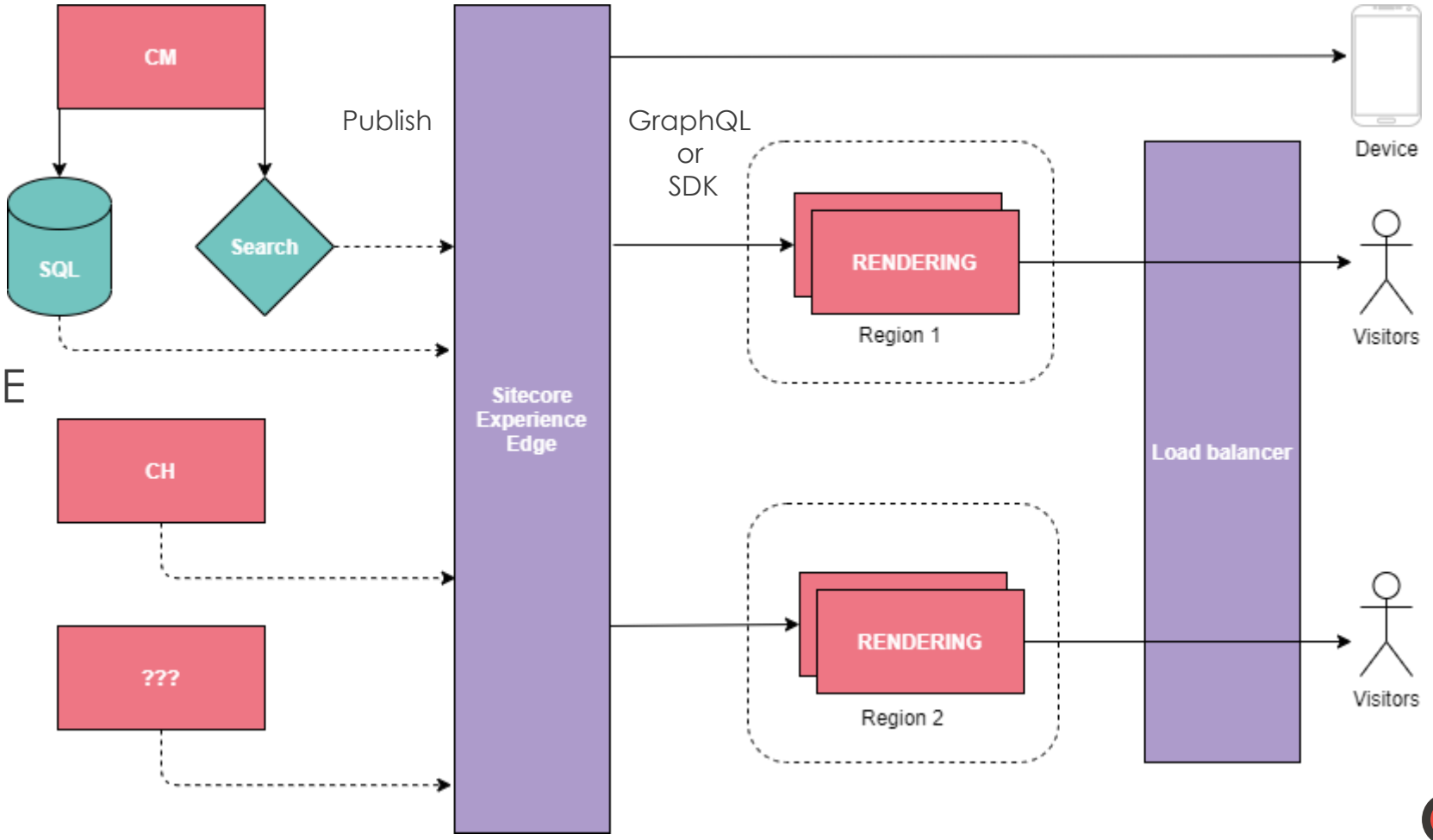
- Content As A Service
- Think CDN + GraphQL

Replaces CD/SQL/Search

- Publish from CM to EE
- RH gets content from EE
  - GraphQL or SDK

Not only for Sitecore XM

- Exposes Content Hub
- Exposes ???



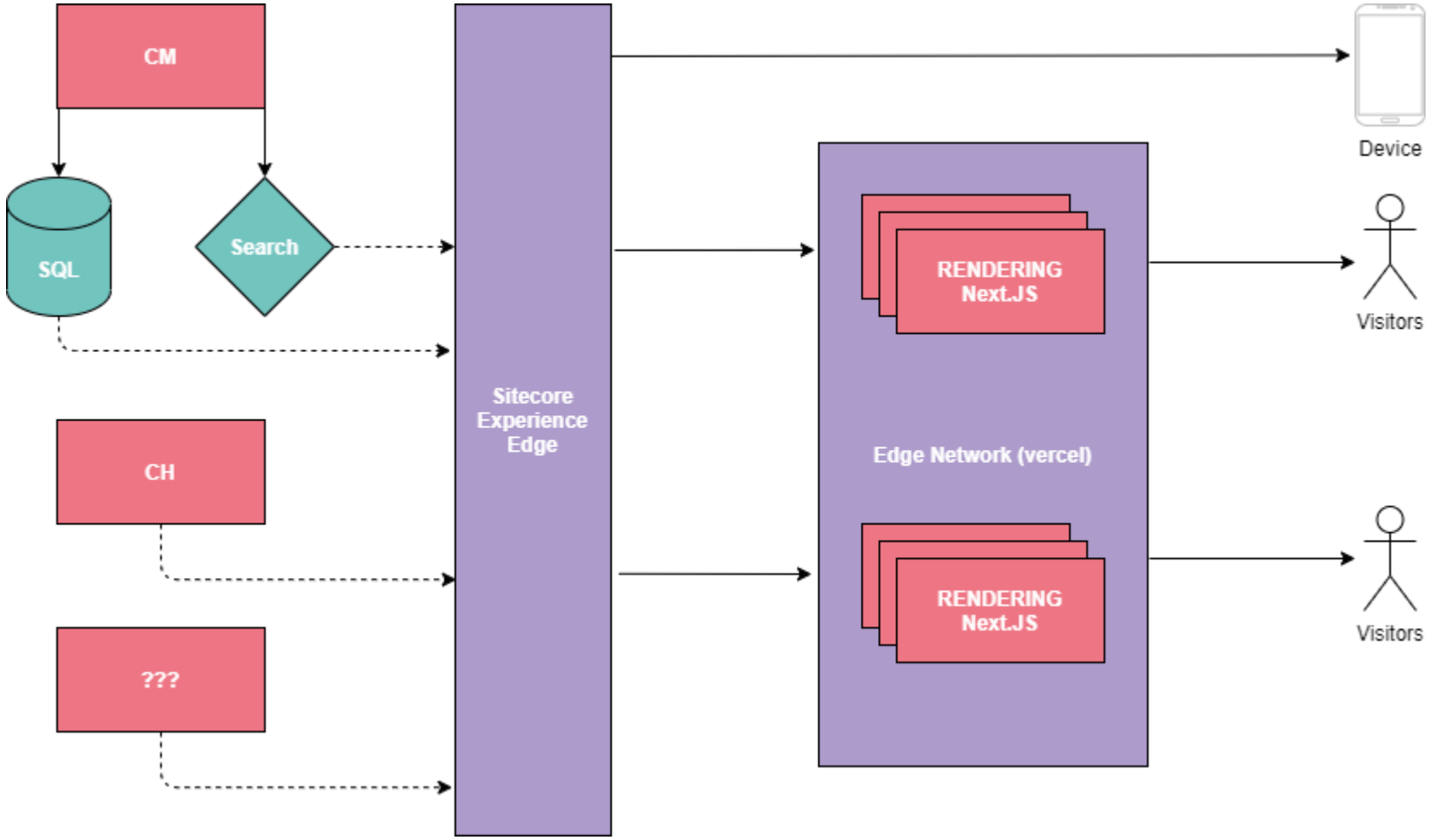
Sitecore Experience Edge w/ Edge network

Rendering on edge network

- Deploy once
- Handles global reach

Examples

- Vercel: Next.JS
  - SSR (Node.JS in edge)
- Azure Static Web Apps
  - CSR
- ???



Demo

How does it work?

- Configure publishing instance
- Illustrate .NET core headless implementation with AI timings across the globe

Possibilities

- Easily switch from .NET core to JSS
- Not just for green field projects: legacy MVC supported (SC10.2)

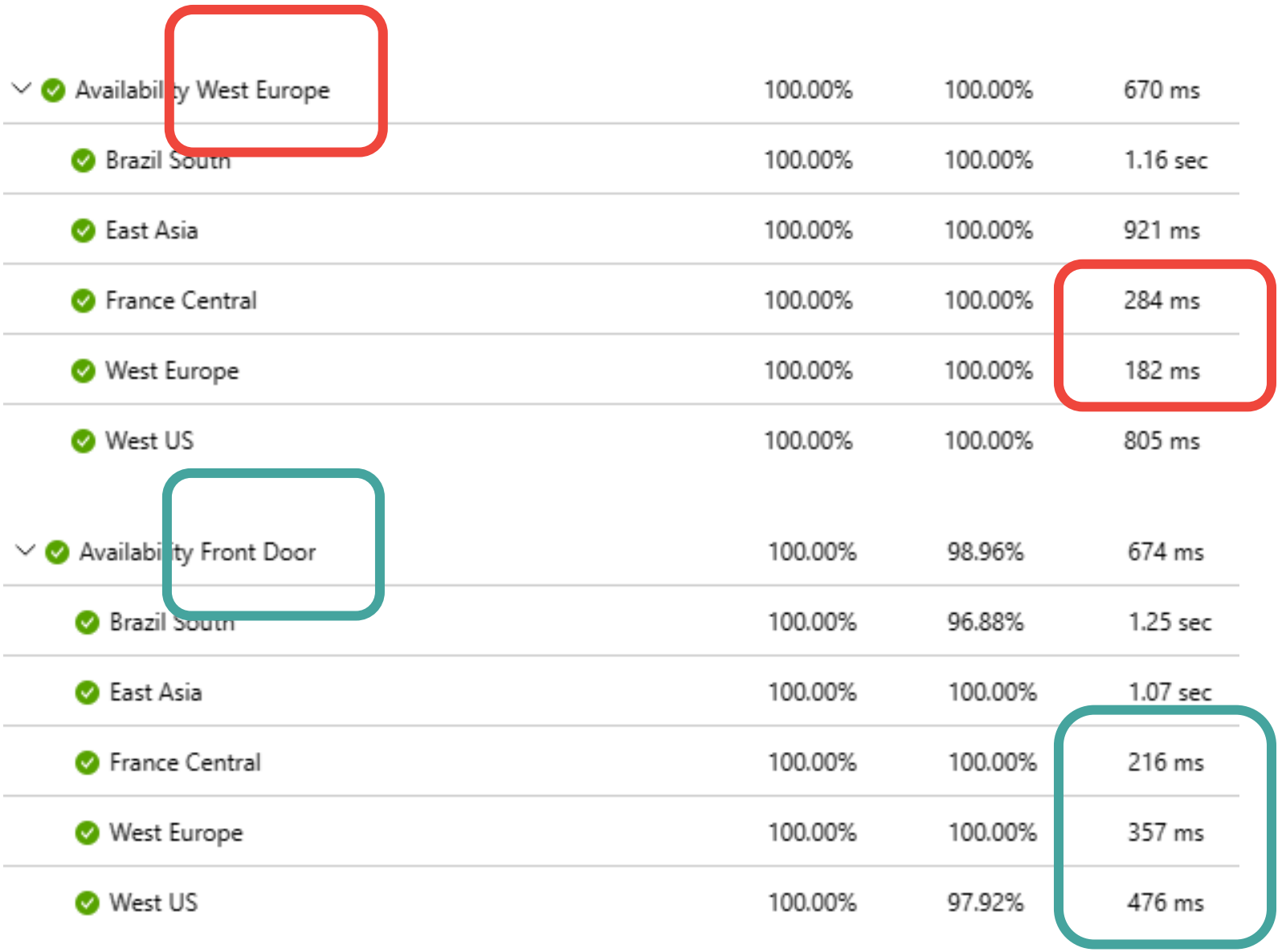
Limitations

- What's not possible?

Demo

Timings

- (No caching)



Availability West Europe	100.00%	100.00%	670 ms
Brazil South	100.00%	100.00%	1.16 sec
East Asia	100.00%	100.00%	921 ms
France Central	100.00%	100.00%	284 ms
West Europe	100.00%	100.00%	182 ms
West US	100.00%	100.00%	805 ms
Availability Front Door	100.00%	98.96%	674 ms
Brazil South	100.00%	96.88%	1.25 sec
East Asia	100.00%	100.00%	1.07 sec
France Central	100.00%	100.00%	216 ms
West Europe	100.00%	100.00%	357 ms
West US	100.00%	97.92%	476 ms

Demo

Limitations (1)

- The EE Connector publishes a static snapshot of the Layout Service output
  - No personalization rules, content testing
  - No dynamic/contextual output (user, querystring)
  - **No enforcement of security (!)**

- No Sitecore analytics/tracking

- Some settings are applied to the whole EE tenant; not per site
  - API Key cannot be restricted to one site
  - Language fallback enabled/disabled

Demo

Limitations (2)

- Media limited to 50MB per item
- Media manipulation limited: h, w, mh, mw

- Publish process works differently
  - "This means that all dynamic content structures in Sitecore (standard values, language fall-backs, rendering data source(s), template inheritance, and so on) are flattened at publishing time. **Therefore, publishing a single item can lead to hundreds of items being published.** For example, if you publish a template item, all items based on that template directly or indirectly […] are also published."

- No easy view on what is currently published (need to use GraphQL item query for that)

Demo

Interesting links

- Install Experience Edge Connector
  https://github.com/nickwesselman/Helix.Examples/blame/aspnet-experienceedge/examples/helix-basic-aspnetcore/docker/build/cm/Dockerfile
- GraphqlLayoutServiceHandler
  https://github.com/nickwesselman/Helix.Examples/tree/aspnet-experienceedge/examples/helix-basic-aspnetcore/src/Feature/ExperienceEdge

- Publishing static output of legacy MVC component
  https://doc.sitecore.com/xp/en/developers/hd/190/sitecore-headless-development/walkthrough--statically-generating-an-mvc-application.html