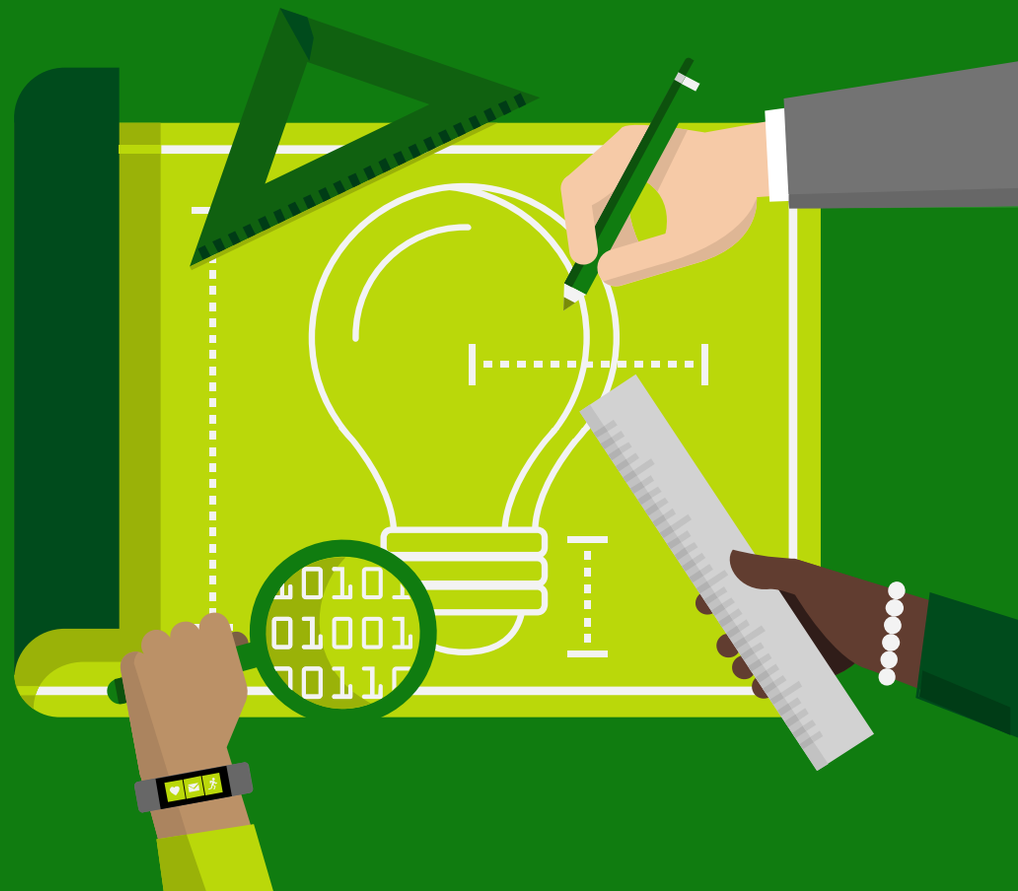
Microsoft

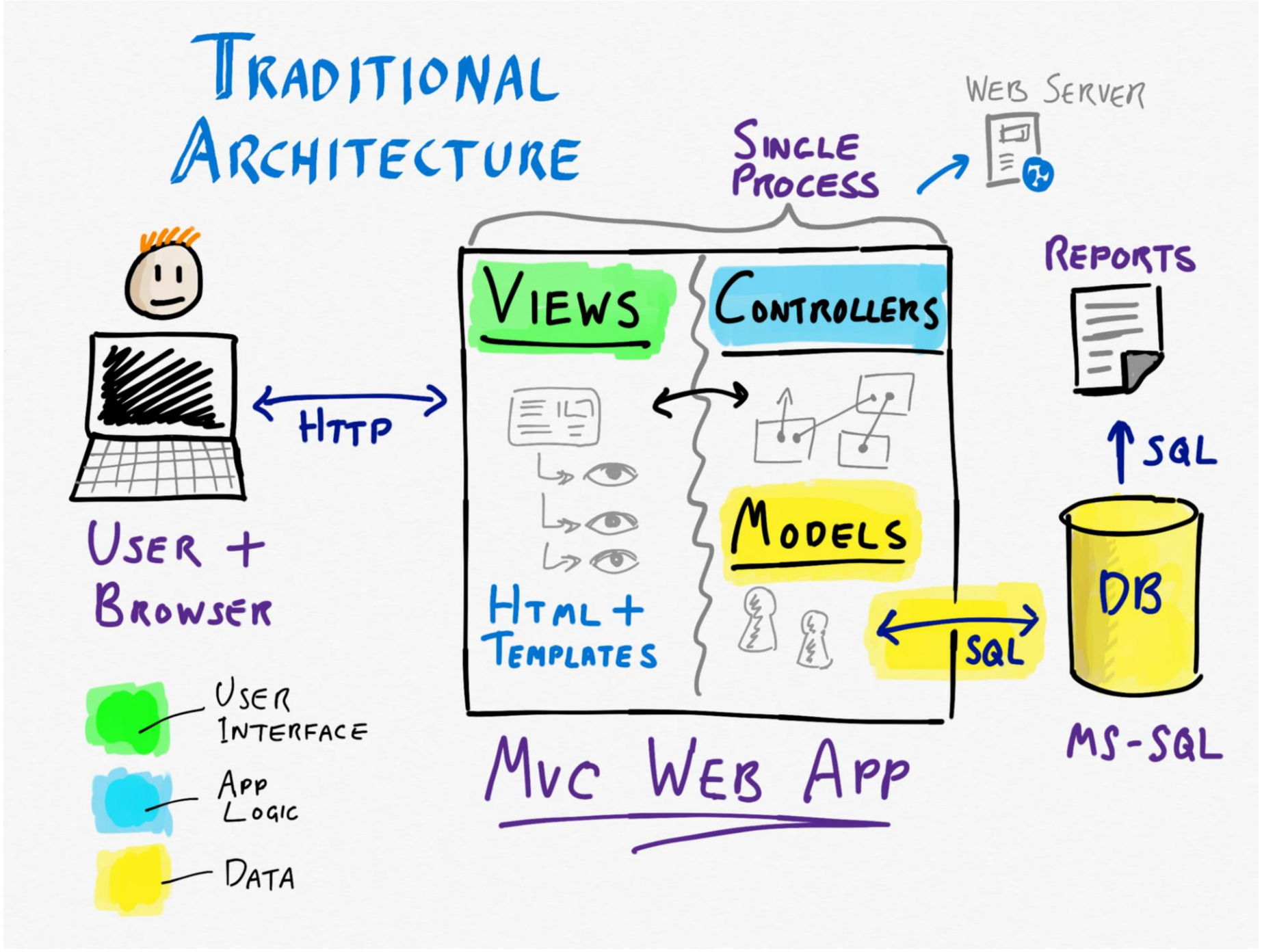
Smilr
Microservices Showcase Application

Ben Coleman
Cloud Architect & Evangelist



Project Goals

- Showcase for Azure compute & app deployment scenarios
- Write & develop a real working system
- Transform a “traditional” application
- Explore use of new design patterns and technologies

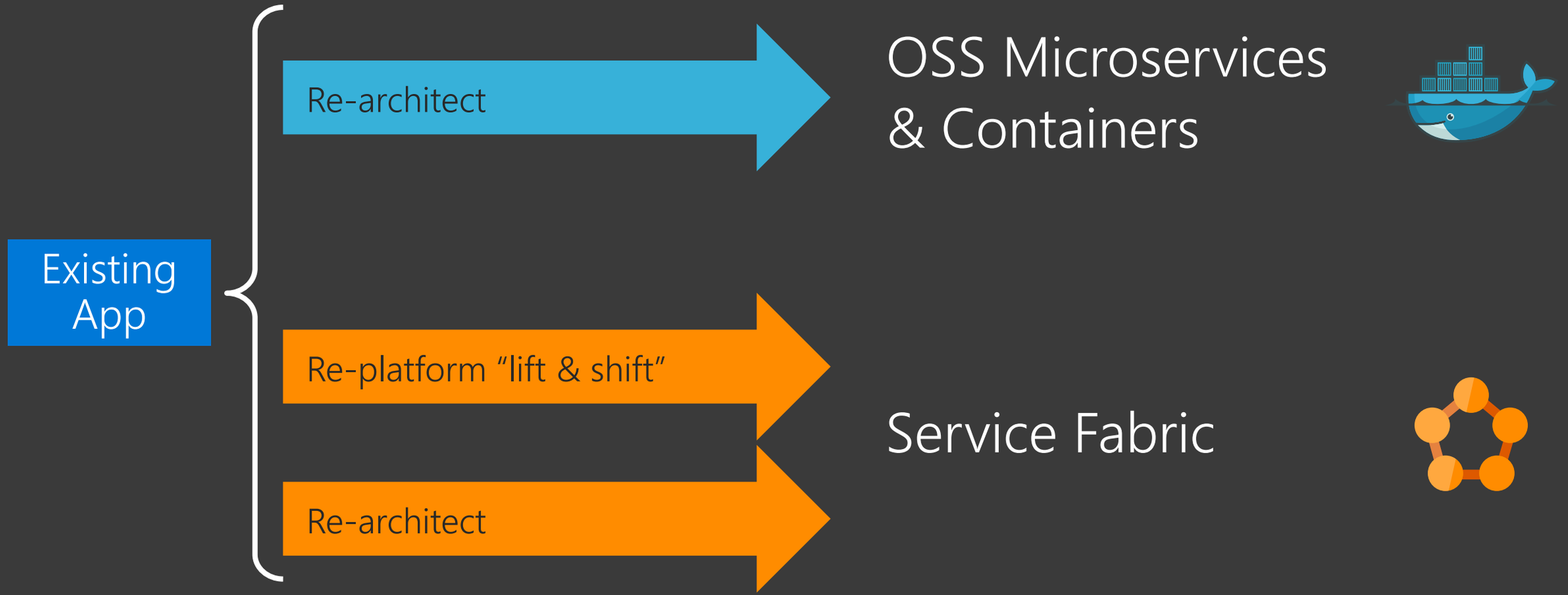


Simple web feedback & rating web app

- Attendees can rate an event (1 to 5 ★)
- Leave comments

Re-platform vs Re-architect

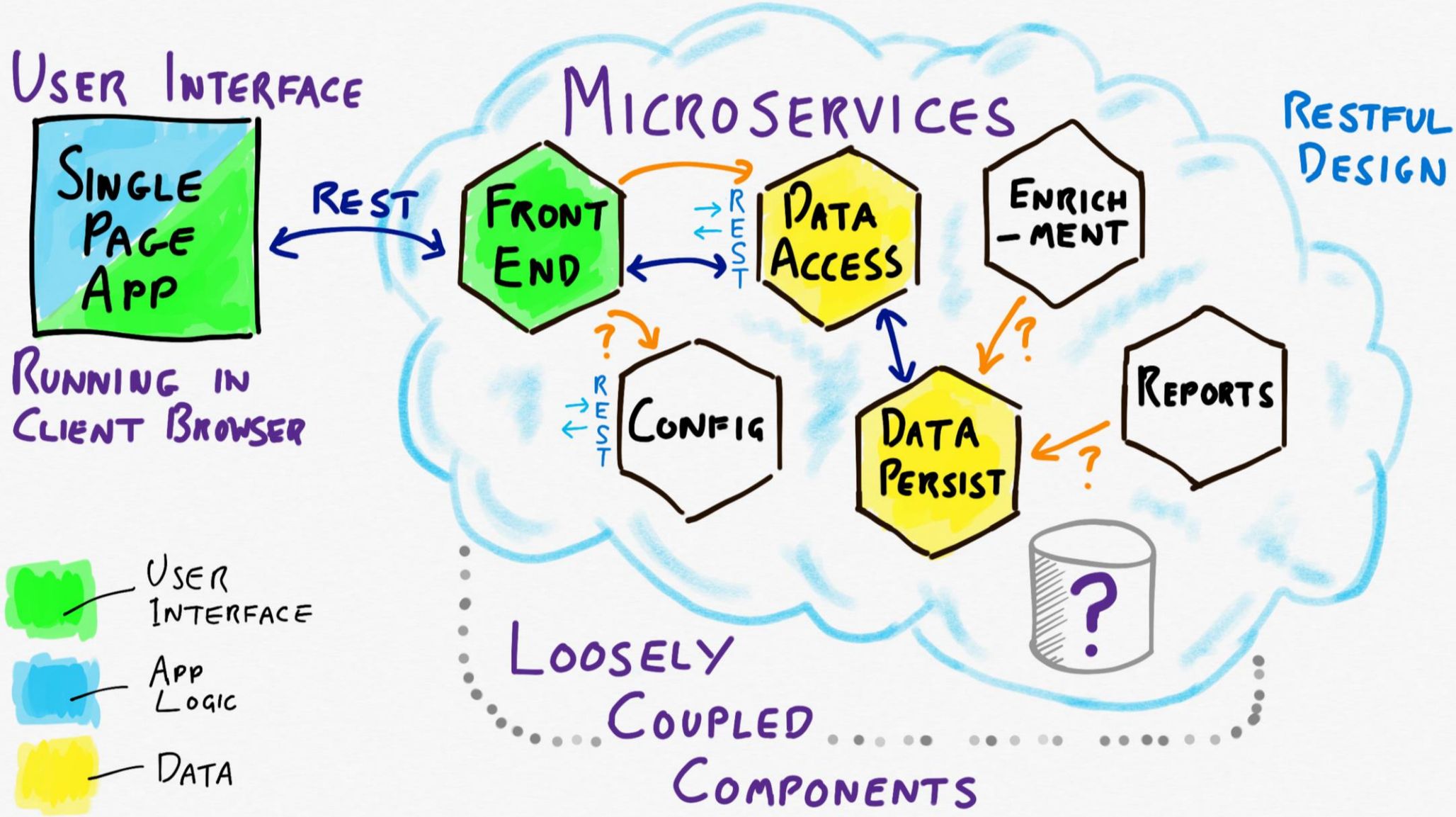
Two technology paths investigated



Re-architect - Technology Goals & Aspirations

- Use loosely coupled microservices
- Complete separation of UI & frontend
  - Single Page Application for UI
- Data store options
- Use of modern lightweight stacks
  - e.g. Node.js & .NET Core
- Use REST based APIs throughout
- Reports & data enrichment

Functional/Concept Architecture



DEMO

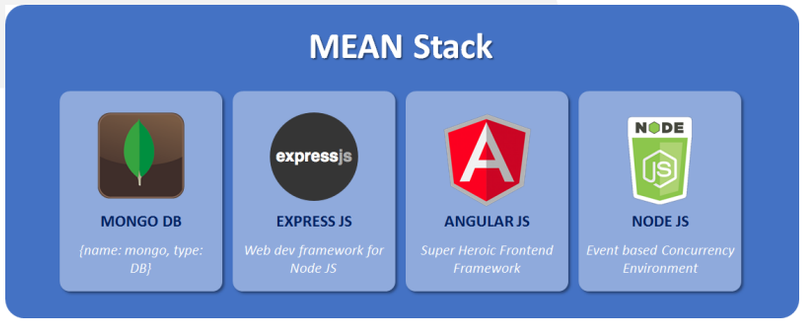
aka.ms/smilr

Microservices - Core Technology Stack

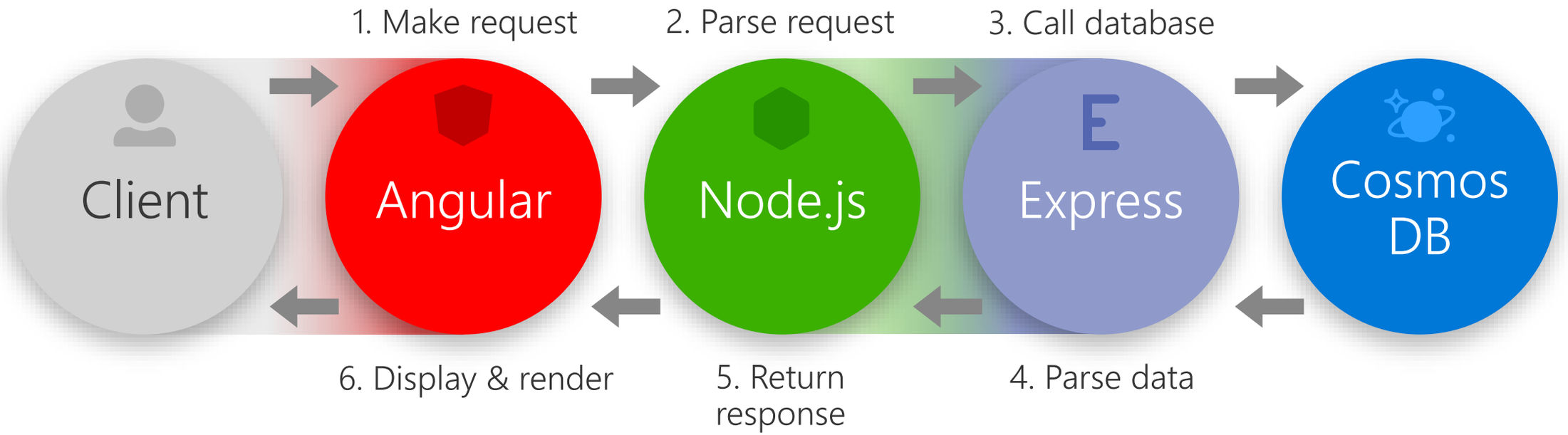
Angular 5 - UI & Client

Node.js & Express - Services & API

Cosmos DB - Database



Basic Data Flow & Interaction



Data Model

ID: w39aZ
"Azure Workshop"
12th October

```
Event {
  id:     any       // Six character UID string or int
  title:  string    // Title of the event, 50 char max
  type:   string    // Type of event ['event', 'workshop', 'hack', 'lab']
  start:  Date      // Start date, an ISO 8601 string; YYYY-MM-DD
  end:    Date      // End date, an ISO 8601 string; YYYY-MM-DD
  topics: Topic[];  // List of Topics, must be at least one
}
```

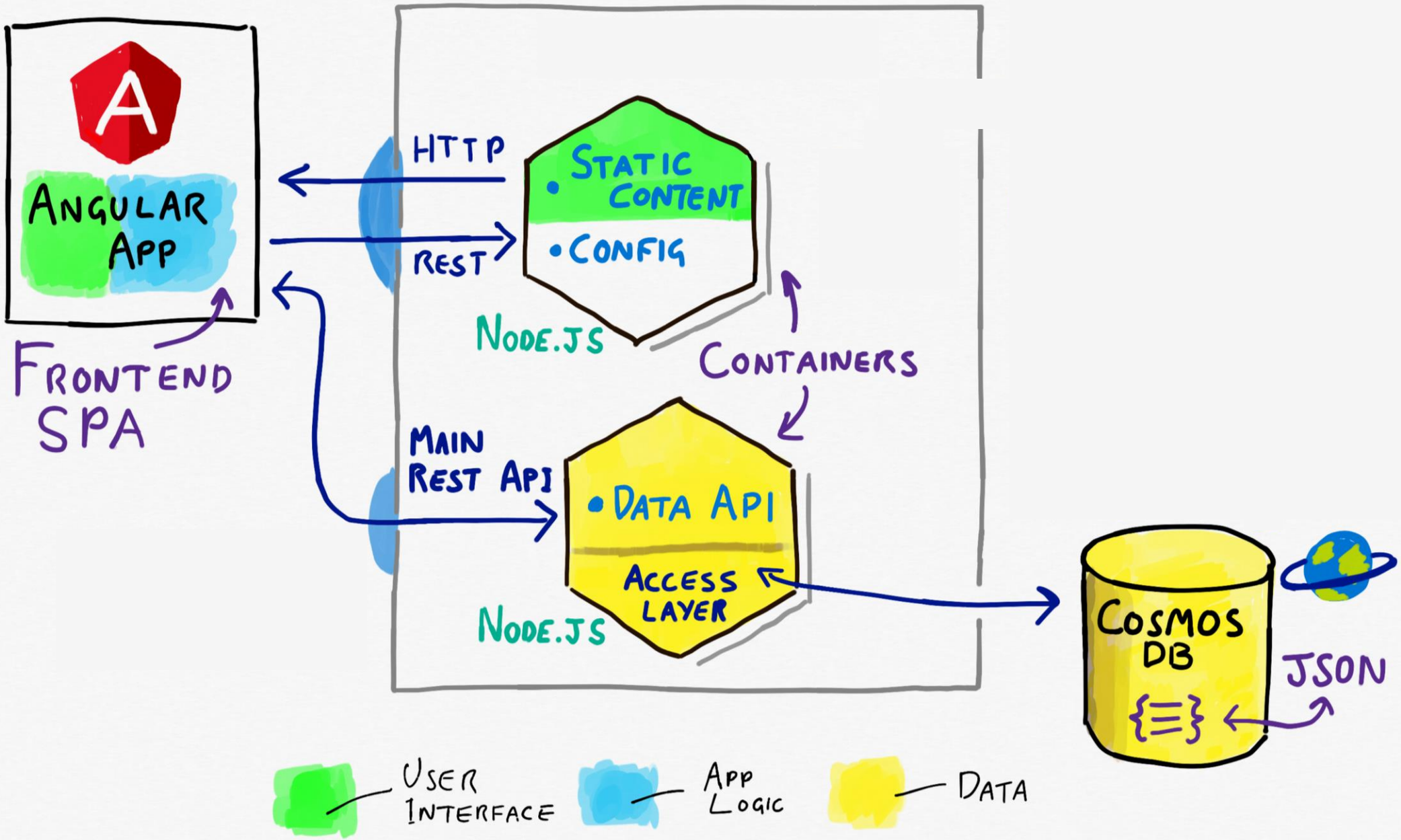
```
Topic {
  id:   number  // Int
  desc: string  // Description
}
```

Topic 1
"Morning Session"

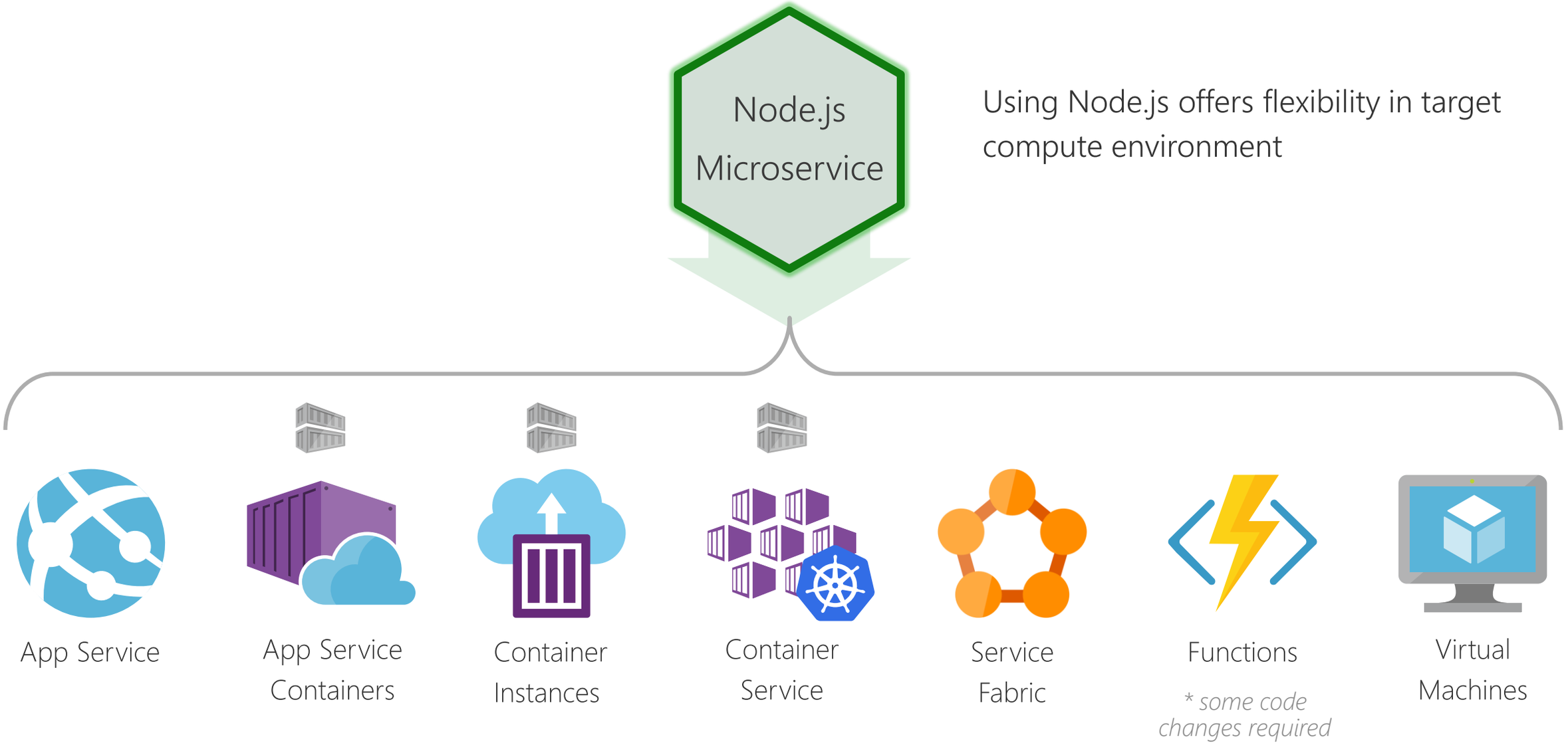
"I learned a lot"
Rating: 4

```
Feedback {
  id:       number  // Six character UID string or int
  event:    string  // Event id
  topic:    number  // Topic id
  rating:   number  // Feedback rating 1 to 5
  comment:  string  // Feedback comments
}
```

Microservices Architecture



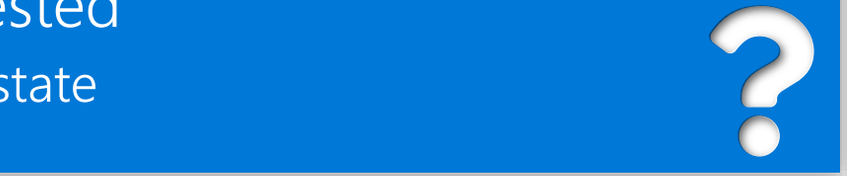
Azure Deployment Options



Using Node.js offers flexibility in target compute environment

Comments - UI / Frontend

- Picked Angular - Over React and Vue.js
  - A fuller framework - less choices required
- Used Angular CLI project structure
- UI Framework - Bootstrap v3, tried and tested
  - Google Material considered - too early/beta state

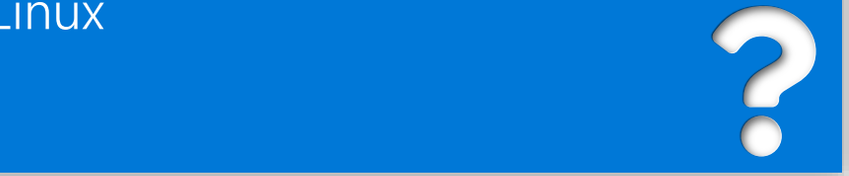


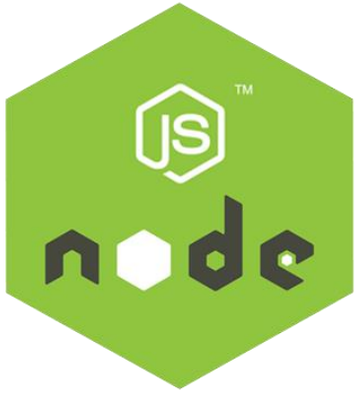
- Move from Angular v4 to v5
- Angular CLI
  - No webpack hell !
  - Hides complexity
- In memory DB & API for rapid development

- Runtime configuration difficult
  - Config service
- Angular.js legacy results when searching
- Docs hit & miss

Comments - Microservices

- Picked Node.js
- Express web framework - Over Koa or Loopback or 100's others!
- Investigated YARN over NPM
  - Fast but issues with Windows Subsystem for Linux



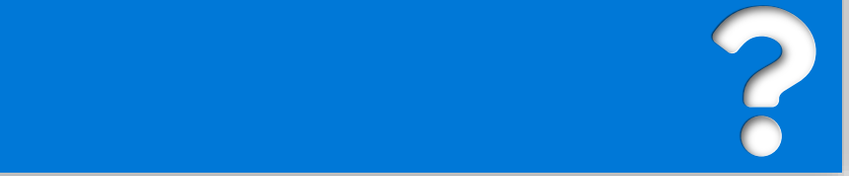


- No issues moving from Node v6 to v8
- Worked on Windows and Linux without change

- Deploying to Azure App Service requires some knowledge & prep

Comments - State & Database

- Initial prototype used Azure Tables
  - JSON serialization, no means to query, no Functions trigger
- Switched to Cosmos DB
- Picked Document DB API
- Mongo was considered

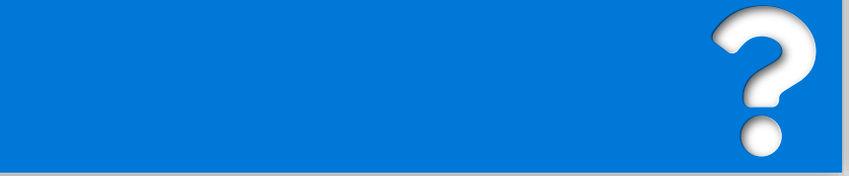


- Cosmos & Doc DB JSON native
  - Made middle tier service simple to write
- API was logical and clean
- Local emulator

- Using multiple collections in Cosmos has cost implications
  - Used partitioned collection
  - Events & feedback in same collection
- Documentation for Doc DB Node SDK was patchy

Comments - Containers

- Docker for Windows 10 - local development
- Azure Container Service (AKS)
- Kubernetes - Over Swarm or DC/OS
- Azure Container Registry (ACR)

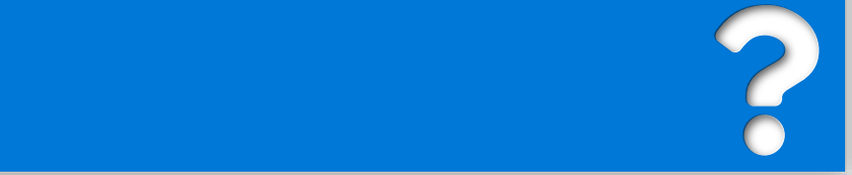


- VS Code creates working Dockerfiles for Node.js projects
- Multi-stage Docker image builds
- ACR works with Docker CLI
- Kubernetes deployment YAML
- Kubernetes networking & LB

- Multiple CLIs (`docker`, `az`, `kubectl`)
- Needed plugins for Azure DNS
  - external-dns.alpha.kubernetes.io

Comments - Serverless

- Azure Functions
- Azure Functions Proxies
- Cognitive Services - Sentiment Analysis



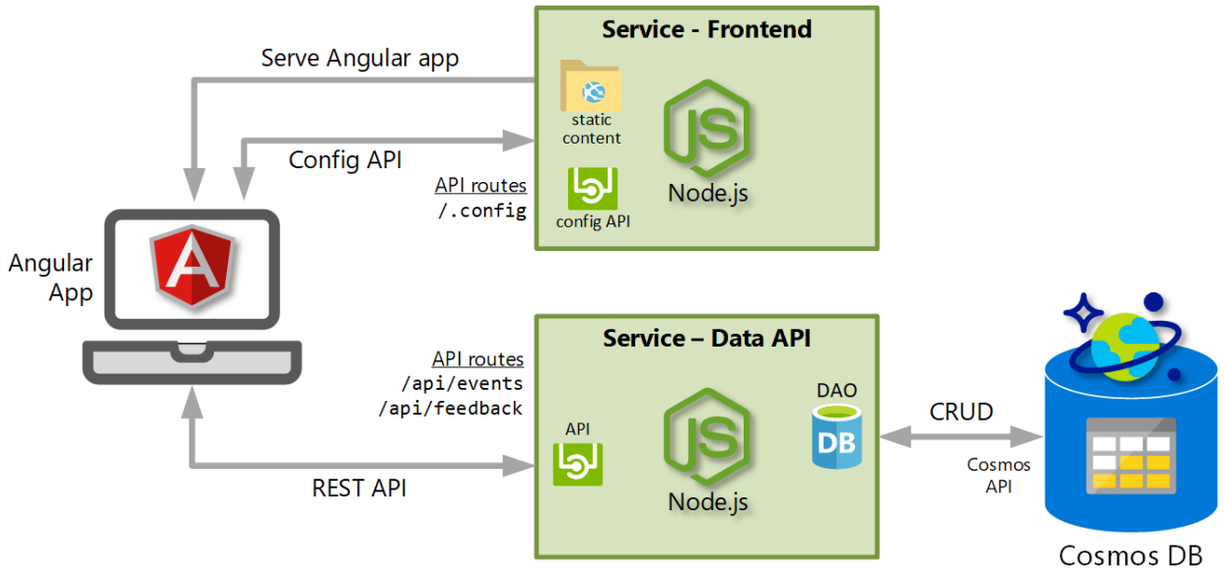
- Swapping Node.js service for Function
  - 100% compatible
  - Dropped in data access library code as-is
  - Some code changes

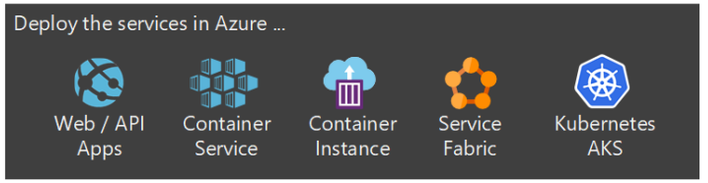
- Cosmos DB trigger for Functions
  - Worked better with C# than Node
  - Infinite document update loop!
- Use Kudu to run NPM install task

Try it out

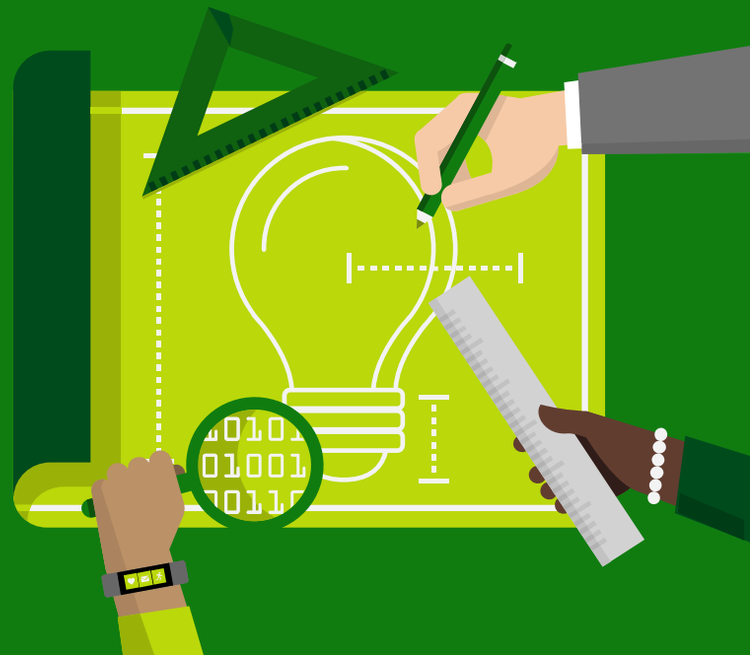
aka.ms/smilr-project

- Guides
- Source code
  - Angular
  - Node services
  - Functions (Node and C#)
- ARM Templates
- Kubernetes deployment





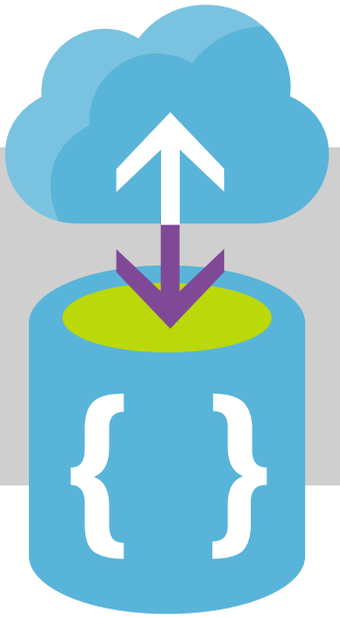
Deeper Dive



Angular - In Memory API

- Dummy API and database for Angular apps
- Allowed for rapid development of frontend design before the real Node REST service(s) were developed
- Not used when deployed in “production” mode

- Represents simple static JSON objects as a fake DB
- Intercepts AJAX (XHR / Fetch) calls and responds like a REST API
- Supports GET, POST, PUT, DELETE



https://github.com/angular/in-memory-web-api

Frontend / UI Service

Just 7 lines of code

- Uses Express
- Simply serves files as static content
- Redirect requests to index.html

```
var express = require('express');
var app = express();

// Serve all static content (index.html, js, css, images, etc.)
app.use('/', express.static(__dirname));

// Redirect all other requests to Angular app index.html,
app.use('*', function(req, res) {
  res.sendFile(`${__dirname}/index.html`);
});

// Start the server
var server = app.listen(process.env.PORT || 3000);
```

https://angular.io/guide/deployment

Frontend / UI Service - Config API

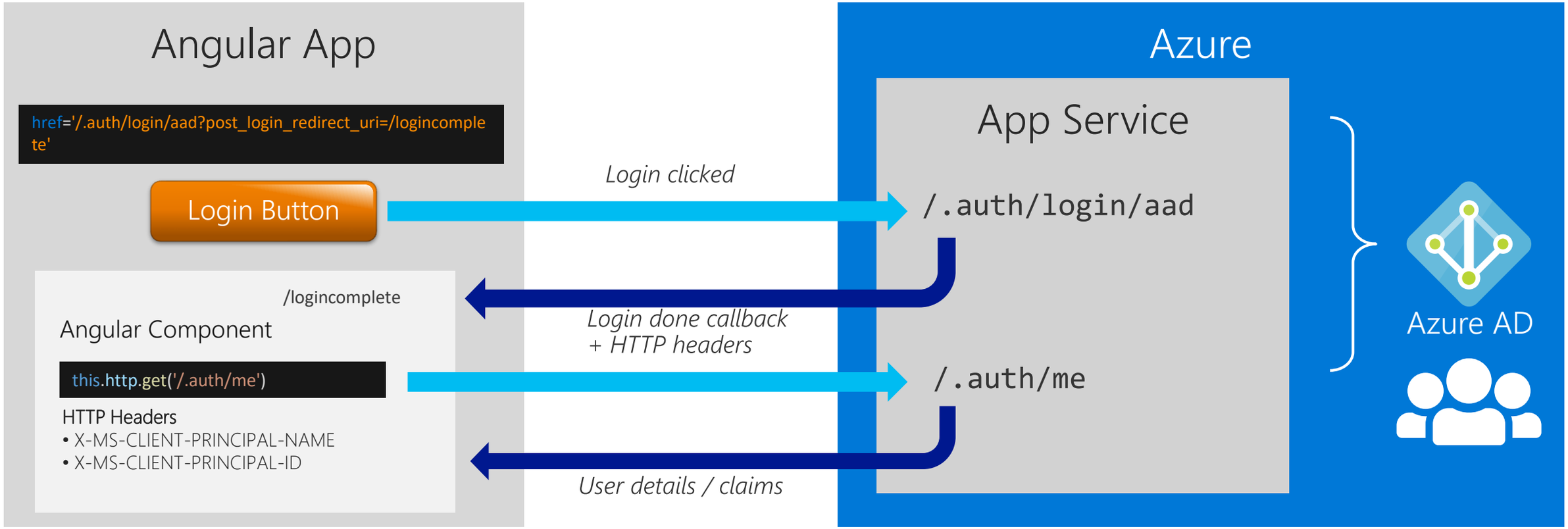
- **Problem** - Angular executes 100% on the client browser.
  How can we dynamically configure settings (e.g. API endpoint address) without code changes and rebuilding?
  (Note. This is a problem for all SPA frameworks, e.g. React, Vue etc)

- **Solution -** Very basic "micro API" on server allowing Angular code to fetch values at startup
  - Lets Angular client side get settings from the server
  - Environmental variables
  - Returns values in JSON

```
//
// MICRO API allowing dynamic configuration of the client/browser side Angular
// Allow Angular to fetch a comma separated set of environmental vars from the server
//
app.get('/.config/:keypairs', function (req, res) {
  let data = {};
  req.params.keypairs.split(",").forEach(varname =>{
    data[varname] = process.env[varname];
  })
  res.send(data);
});
```

Authorization

- Enabled with feature toggle in Angular app
- Requires front-end service hosted in Azure App Service
- Uses turn key 'App Service Authorization' feature
- Enabled, but anonymous access allowed




Microsoft

Future Plans / Roadmap

- Turn frontend UI into progressive app - work offline
- Port Node.js parts to .NET Core