Microservices from Real Life

Mladen Stojanovic, 2017

Why architecture talk on JavaScript meetup?

- for every platform/technology
- broadening your knowledge in the area of system architecture and operations
- brings you further as professional and makes your code better
- ... and there will be JavaScript stuff

Microservice Architecture

Microservices is a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services should be fine-grained and the protocols should be lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity and makes the application easier to understand, develop and test. It also parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently.[1] It also allows the architecture of an individual service to emerge through continuous refactoring.[2] Microservices-based architectures enable continuous delivery and deployment.[3]

Source:
https://en.wikipedia.org/wiki/Microservices

Dooer Microservice Framework

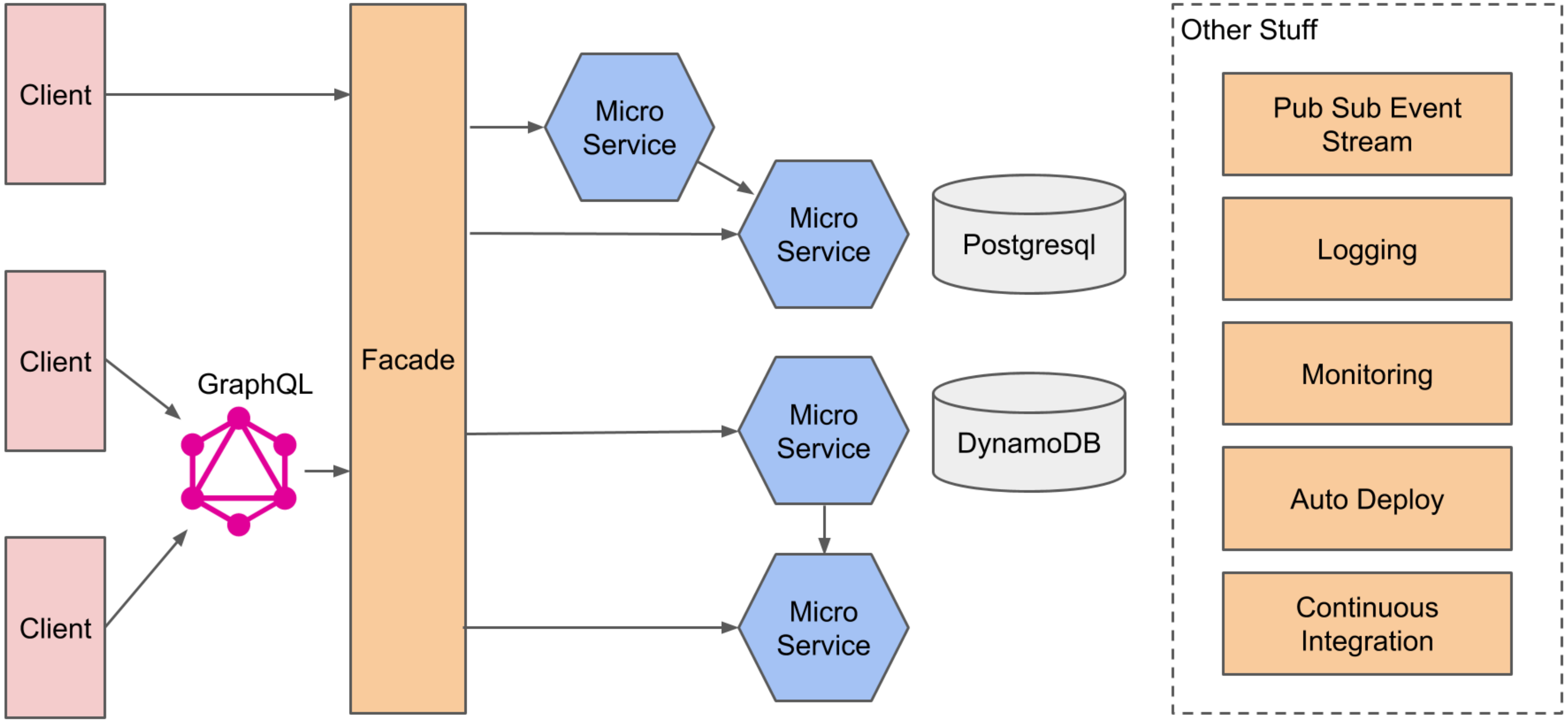
Some numbers:

- 330 github repos
- 40+ (micro) services
- 268 npm packages

Technology:

- React
- Node.js
- Postgresql
- AWS

Architecture



Challenges

- Building Homogenic System
- Documentation
- Service Discovery
- Storage for Secrets
- Automated Deployment

Building Homogenic System

- Microservice Framework
- Frontend Framework
- Frontend Components

Microservice Framework:

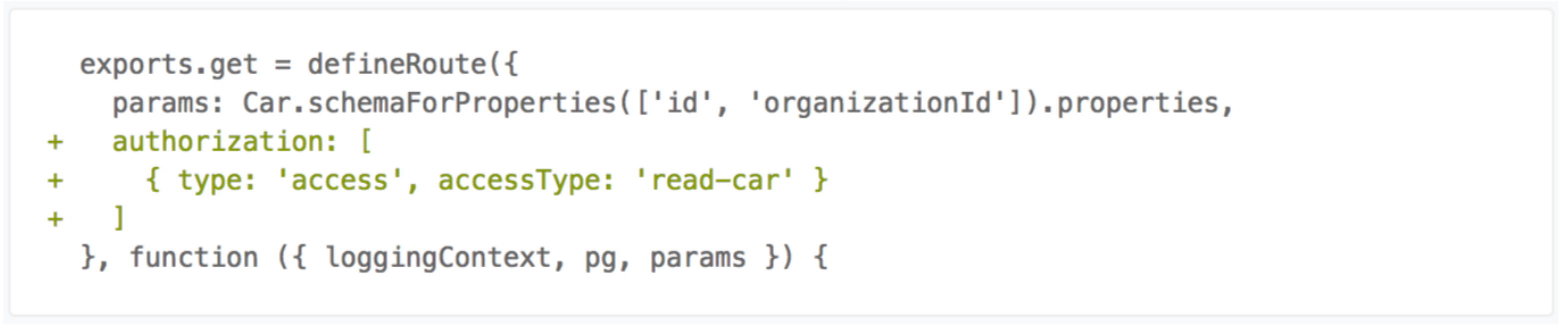
- Express (node.js)
- Authorization
- Documentation
- Logging
- Error Handling
- API Stuff
  - Authentication
  - Pagination
  - Filtering DQL
  - Rate Limiting
  - Mandatory Headers
  - Metrics Reporting
  - Other stuff (naming standards, data formats, etc.)

Frontend Framework:

- React
- Supported Browsers
- View State
- Application State
  https://github.com/jumpsuit/jumpstate
- Internationalization (gettext)
- Validation and Normalization

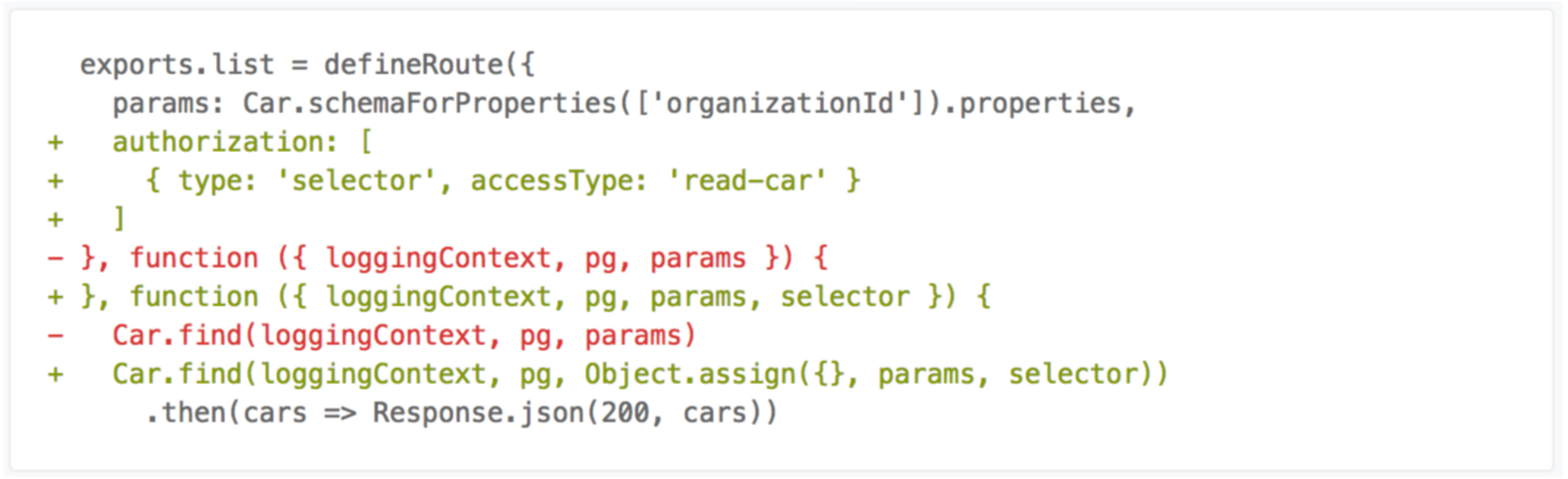
Microservice Framework / Authorization

Require access for a specific resource



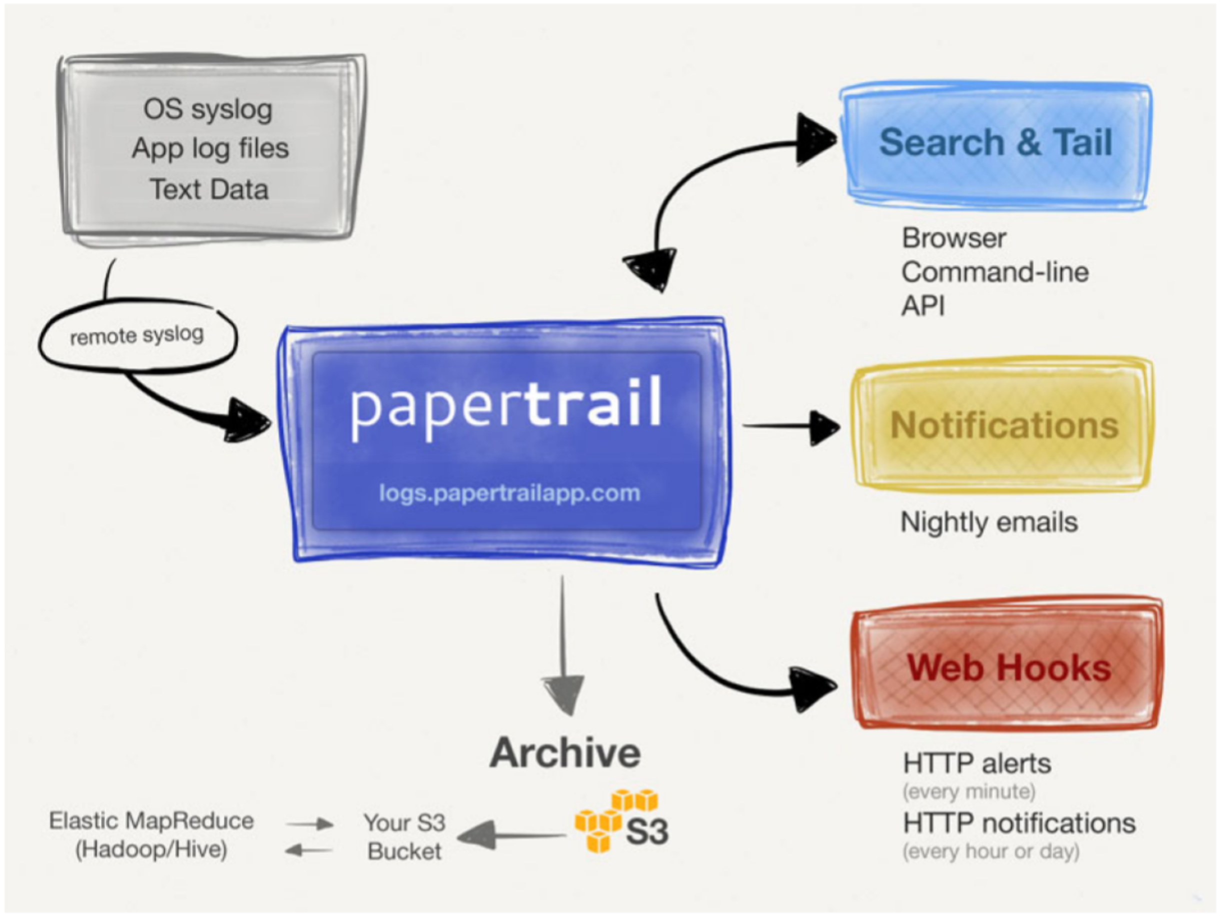
```
  exports.get = defineRoute({
    params: Car.schemaForProperties(['id', 'organizationId']).properties,
+   authorization: [
+     { type: 'access', accessType: 'read-car' }
+   ]
  }, function ({ loggingContext, pg, params }) {
```

Listing resources with required access



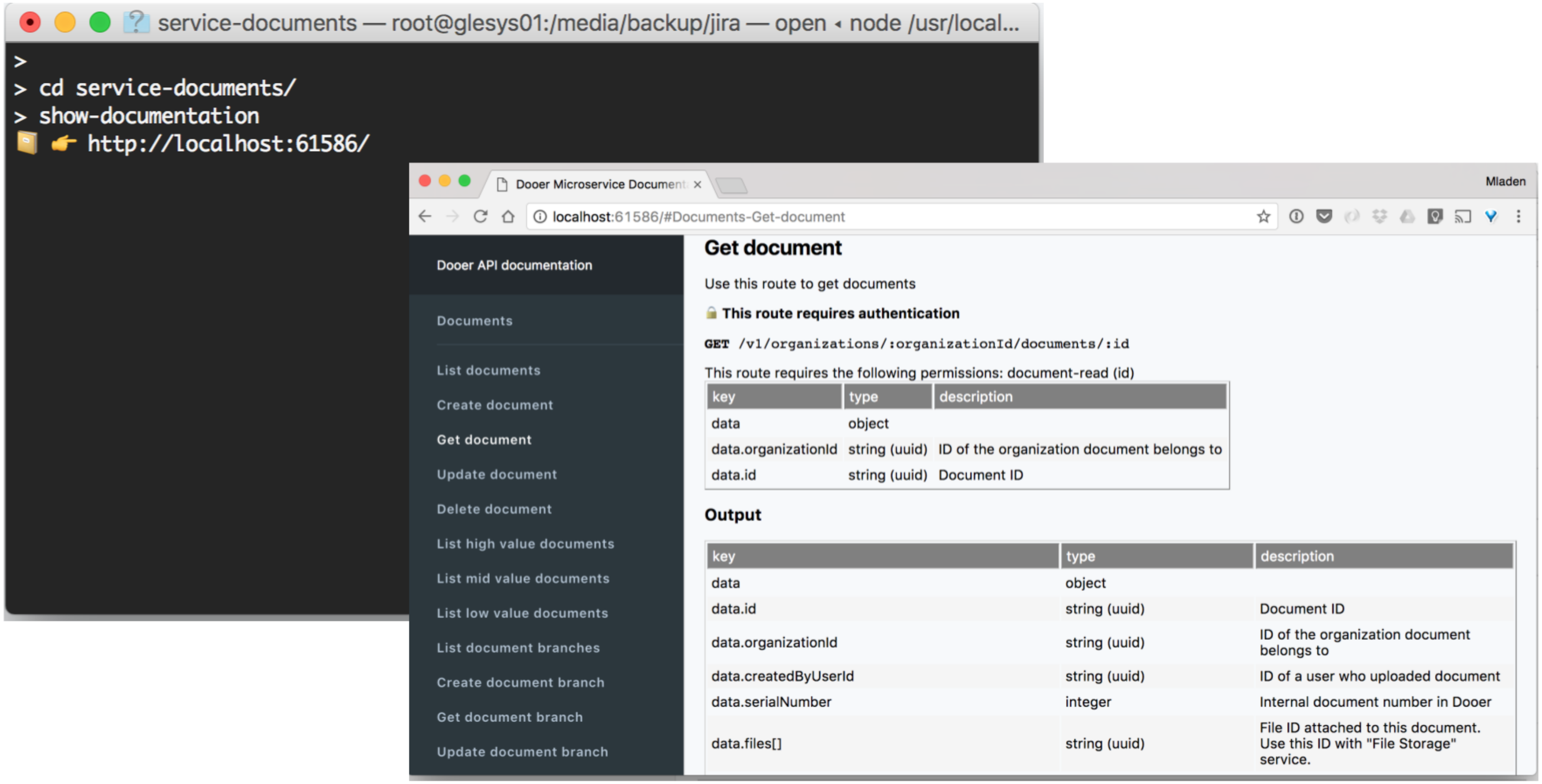
```
  exports.list = defineRoute({
    params: Car.schemaForProperties(['organizationId']).properties,
+   authorization: [
+     { type: 'selector', accessType: 'read-car' }
+   ]
- }, function ({ loggingContext, pg, params }) {
+ }, function ({ loggingContext, pg, params, selector }) {
-   Car.find(loggingContext, pg, params)
+   Car.find(loggingContext, pg, Object.assign({}, params, selector))
      .then(cars => Response.json(200, cars))
```

Microservice Framework / Logging



Source: https://papertrailapp.com/

Microservice Framework / Documentation



service-documents — root@glesys01:/media/backup/jira — open ◂ node /usr/local...

```
>
> cd service-documents/
> show-documentation
📔 👉 http://localhost:61586/
```

Get document

Use this route to get documents

🔒 **This route requires authentication**

```
GET /v1/organizations/:organizationId/documents/:id
```

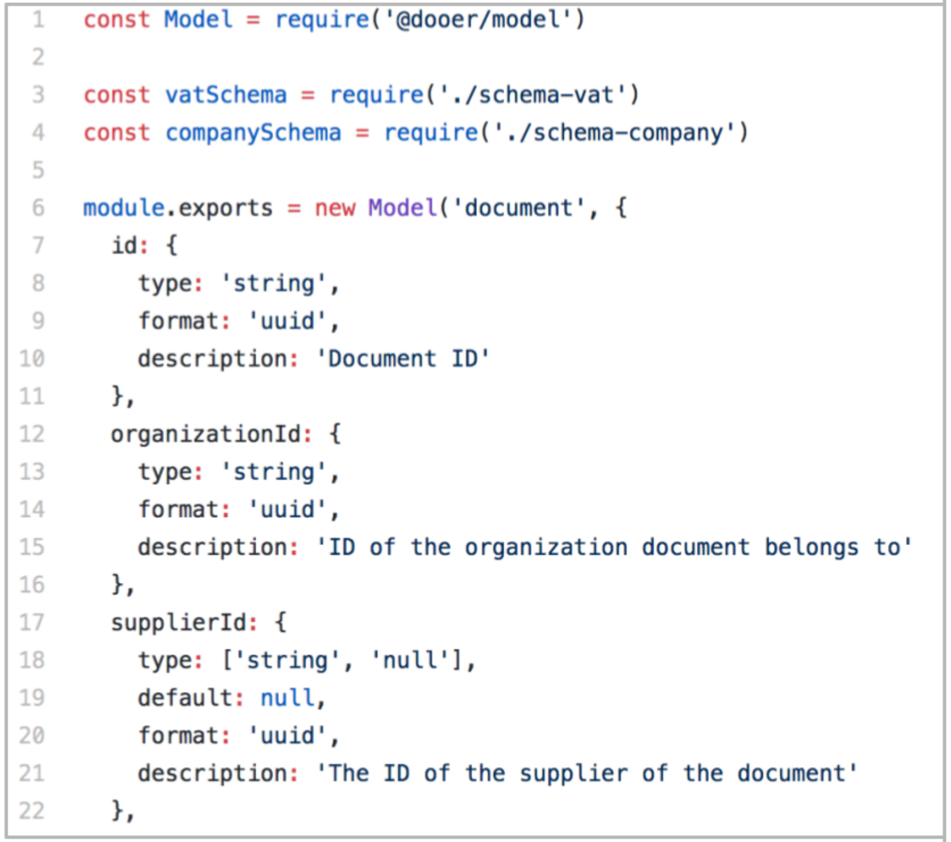
This route requires the following permissions: document-read (id)

key	type	description
data	object	
data.organizationId	string (uuid)	ID of the organization document belongs to
data.id	string (uuid)	Document ID

Output

key	type	description
data	object	
data.id	string (uuid)	Document ID
data.organizationId	string (uuid)	ID of the organization document belongs to
data.createdByUserId	string (uuid)	ID of a user who uploaded document
data.serialNumber	integer	Internal document number in Dooer
data.files[]	string (uuid)	File ID attached to this document. Use this ID with "File Storage" service.

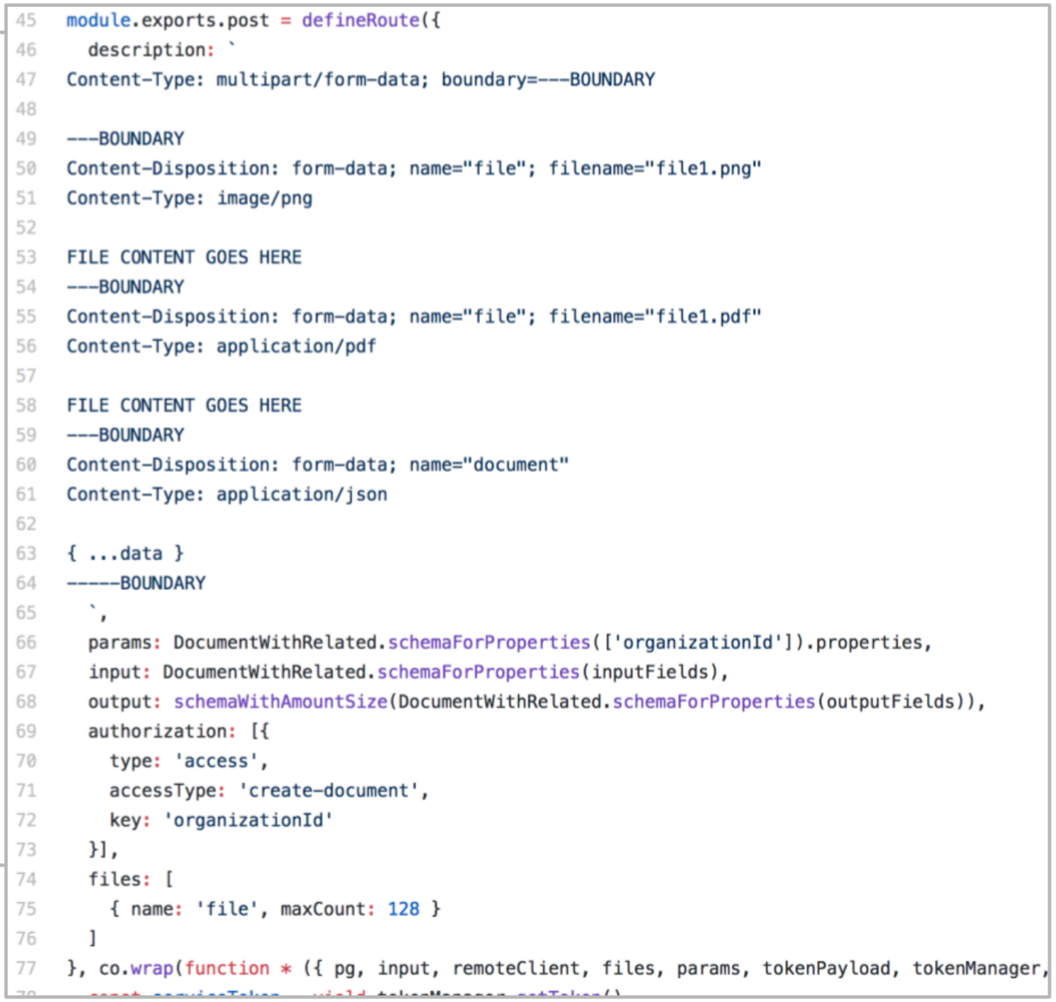
Documentation - directly from the code



```js
const Model = require('@dooer/model')

const vatSchema = require('./schema-vat')
const companySchema = require('./schema-company')

module.exports = new Model('document', {
  id: {
    type: 'string',
    format: 'uuid',
    description: 'Document ID'
  },
  organizationId: {
    type: 'string',
    format: 'uuid',
    description: 'ID of the organization document belongs to'
  },
  supplierId: {
    type: ['string', 'null'],
    default: null,
    format: 'uuid',
    description: 'The ID of the supplier of the document'
  },
```



```js
module.exports.post = defineRoute({
  description: `
Content-Type: multipart/form-data; boundary=---BOUNDARY

---BOUNDARY
Content-Disposition: form-data; name="file"; filename="file1.png"
Content-Type: image/png

FILE CONTENT GOES HERE
---BOUNDARY
Content-Disposition: form-data; name="file"; filename="file1.pdf"
Content-Type: application/pdf

FILE CONTENT GOES HERE
---BOUNDARY
Content-Disposition: form-data; name="document"
Content-Type: application/json

{ ...data }
-----BOUNDARY
  `,
  params: DocumentWithRelated.schemaForProperties(['organizationId']).properties,
  input: DocumentWithRelated.schemaForProperties(inputFields),
  output: schemaWithAmountSize(DocumentWithRelated.schemaForProperties(outputFields)),
  authorization: [{
    type: 'access',
    accessType: 'create-document',
    key: 'organizationId'
  }],
  files: [
    { name: 'file', maxCount: 128 }
  ]
}, co.wrap(function * ({ pg, input, remoteClient, files, params, tokenPayload, tokenManager,
```

HashiCorp

- Service Discovery
- Storage for Secrets
- Automated Deployment

HashiCorp Terraform

HashiCorp Nomad

HashiCorp Consul

HashiCorp Vault

HashiCorp

- Service Discovery
- Storage for Secrets
- Automated Deployment

HashiCorp Terraform

HashiCorp Nomad

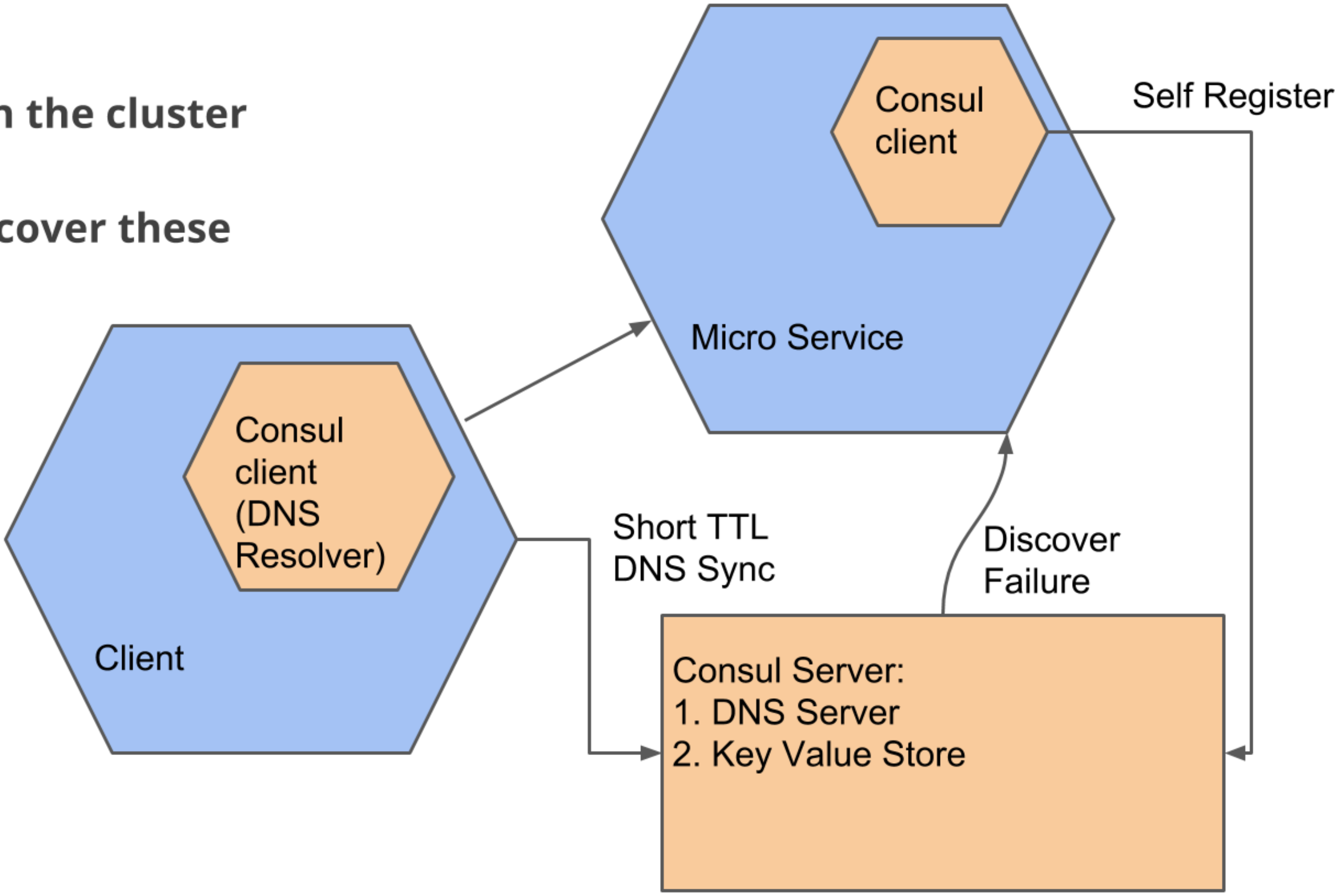
HashiCorp Consul

HashiCorp Vault

Service Discovery

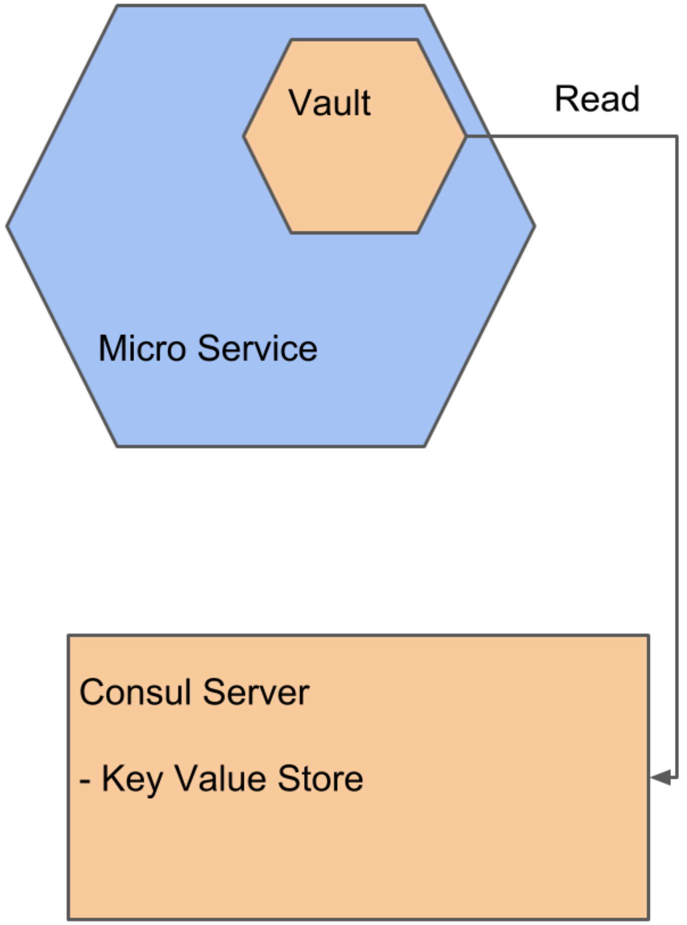
- New Service
- Service changes host
- Service gets more nodes in the cluster

- How do service clients discover these changes?

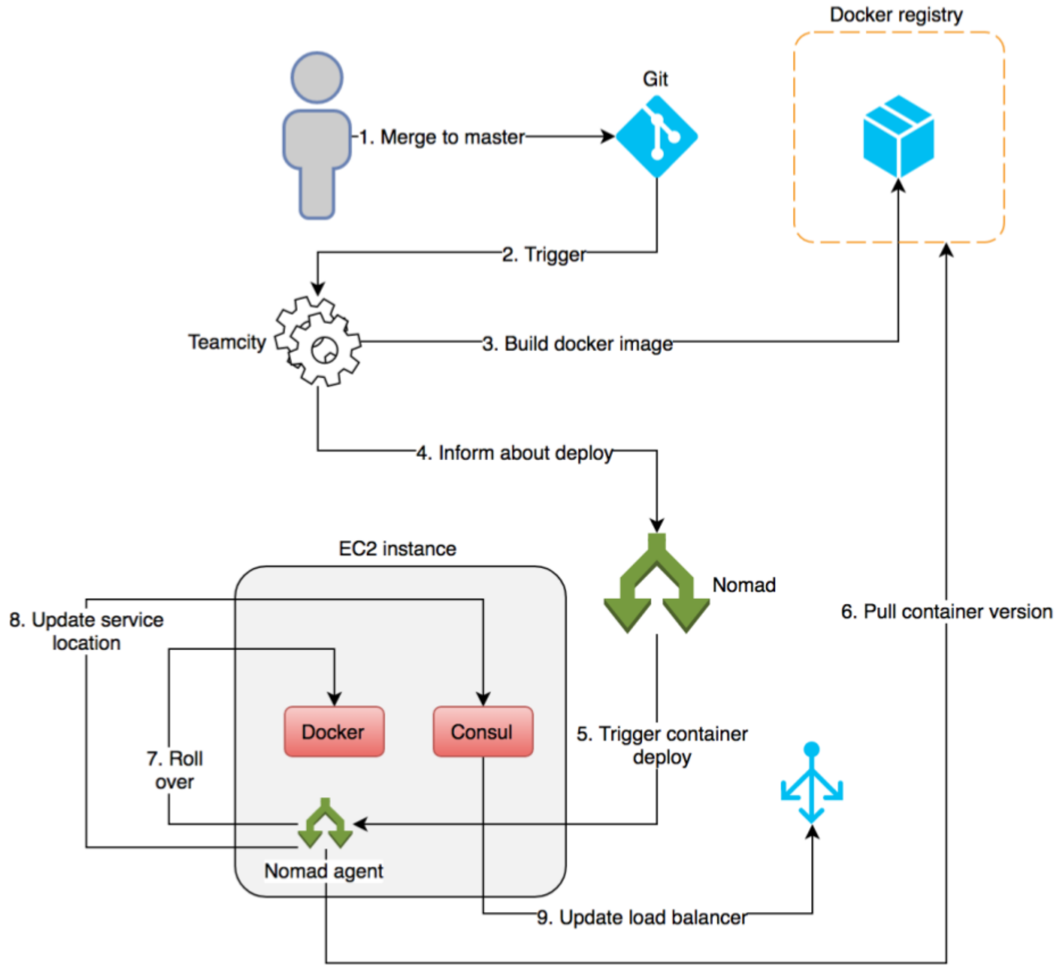


Storage for Secrets

- Storage for secrets
- Key Rolling
- Audit Log



Deploy One Microservice



Experience

- Expensive to set up,
- but it really pays off (scalability, overall "neatness"),
- ONLY IF you need it (super-scalable, super-decoupled)

- Feature-discipline

- Automation is MUST HAVE
  - Deploy in minutes
  - Setup dev environment in minutes
  - Documentation
  - CI, etc.

- Agile++ really (and only) worked

- Best practices still emerging
  http://www.vinaysahni.com/best-practices-for-building-a-microservice-architecture
  https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/

Thank you!