OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

WEBINAR

Single Page Apps for a Microservices Architecture

Presented by Zachary Klein
Senior Software Engineer

MICRONAUT

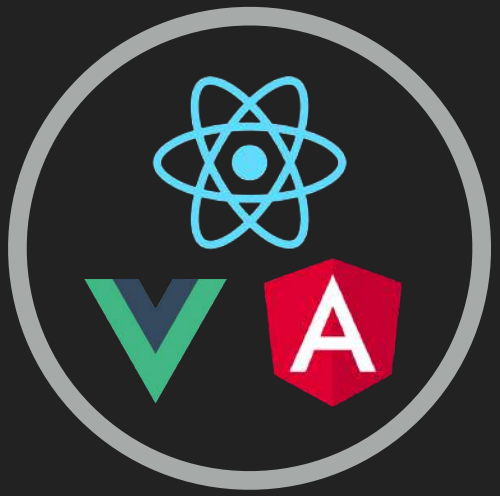
About Me

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

- Zachary Klein
- Senior Software Engineer
- “Full Stack!”
- OSS contributor to Grails and Micronaut
- Twitter: @ZacharyAKlein

AGENDA

- Brief Introduction to Micronaut
- RESTful Backends with Micronaut
- API Gateways
- Security with JWT
- Token Propagation
- Multi-tenancy



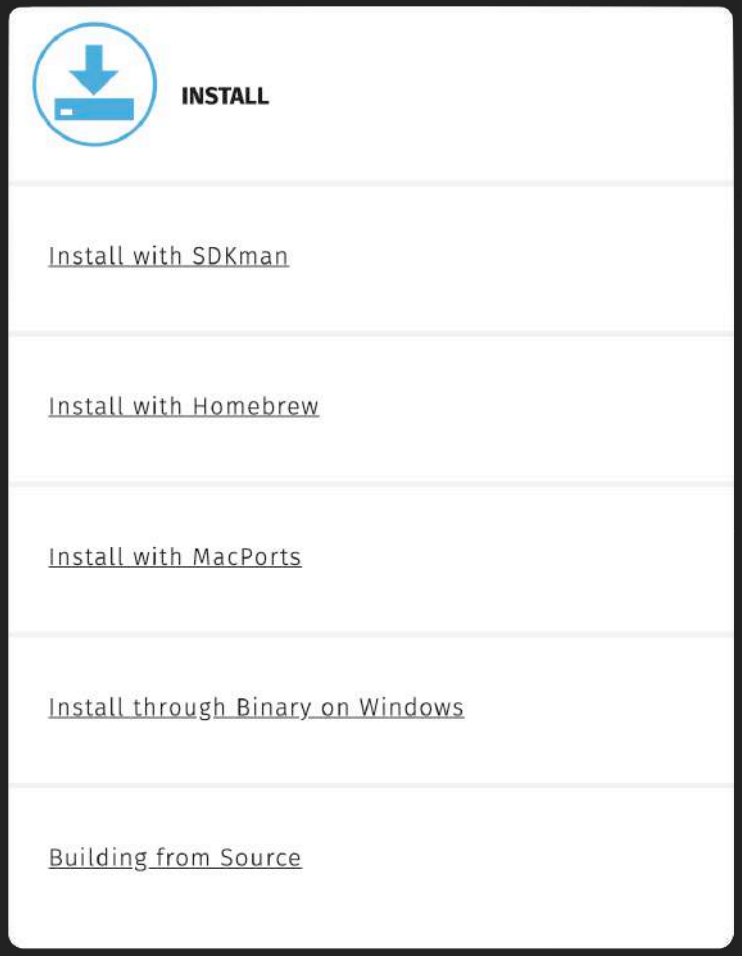
BRIEF INTRODUCTION TO MICRONAUT

MICRONAUT

- Designed with Microservices in mind
- Reactive HTTP server built on Netty
- AOT (Ahead of Time) Compilation for DI, AOP, and configuration
- Declarative HTTP Client
- “Natively” Cloud-Native: service-discovery, load-balancing, circuit-breakers, tracing, and more!
- Support for Java, Kotlin and Groovy

MICRONAUT: GETTING STARTED

```
~ curl -s "https://get.sdkman.io" | bash
~ source "$HOME/.sdkman/bin/sdkman-init.sh"
~ sdk install micronaut
~ mn create-app hello-world
```



https://micronaut.io/download.html

MICRONAUT CLI

- Language defaults to Java
  - Use `--lang` to specify **groovy** or **kotlin**
- Build tool defaults to Gradle
  - Use `--build` to specify **maven**
- Run `mn` without arguments to enter interactive mode
  - includes tab-completion

MICRONAUT: CONTROLLERS & CLIENTS

```
@Controller("/")
class HelloController {

  @Get("/hello/{name}")
  String hello(String name) {
    return "Hello " + name;
  }

}
```

```
@Client("/")
interface HelloClient {

  @Get("/hello/{name}")
  String hello(String name);

  // Implementation generated
  // at compile time
}
```

MICRONAUT: DEPENDENCY INJECTION

```
@Singleton //Bean definition generated at compile time
class WeatherService {
  Integer currentTemp() { //... }
}

@Controller('/weather')
class WeatherController {

    @Inject WeatherService weatherService
    //DI computed at compile time

    @Get("/")
    Integer currentTemp() {
      return weatherService.currentTemp()
    }
}
```

MICRONAUT: CLOUD NATIVE

SERVICE DISCOVERY

```
//Lookup client from service-discovery registry
@Client(id="billing", path="/billing")
interface BillingClient { ... }
```

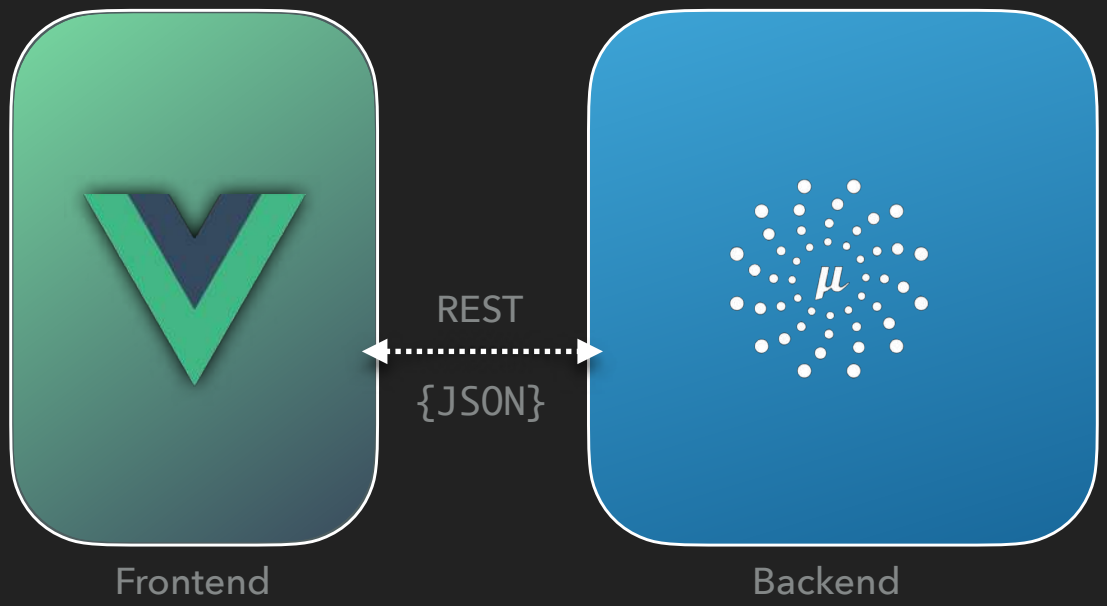
RETRYABLE

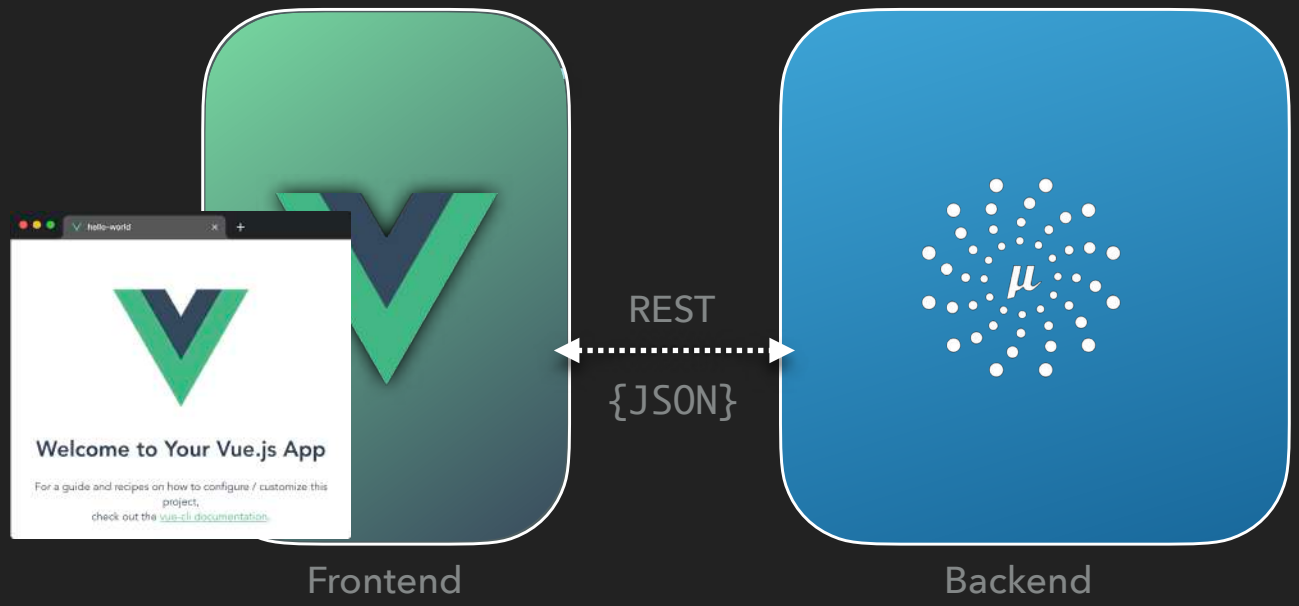
```
//Automatically retry failing calls
@Client("https://api.external.service")
@Retryable(attempts = '3', delay = '5ms')
interface ExternalApiClient { ... }
```

CIRCUIT BREAKERS

```
//Immediately fail after set number of failures
//Begin accepting calls after `reset` interval
@Singleton
@CircuitBreaker(attempts = '5', reset = '300ms')
class MyService { ... }
```

RESTFUL BACKENDS WITH MICRONAUT


REST
{JSON}
Frontend
Backend


hello-world
Welcome to Your Vue.js App
For a guide and recipes on how to configure / customize this project,
check out the vue-cli documentation.
REST
{JSON}
Frontend
Backend

MICRONAUT & REST

- Declarative Routes via method annotations:
  - `@Get`, `@Put`, `@Post`, `@Delete`
- JSON binding/rendering via Jackson
- Request Arguments via annotations:
  - `@Header`, `@Body`, `@CookieValue`, `@QueryValue`

JACKSON: JSON BINDING

```
public class Author {

    private String name;

    @JsonSerialize(MySerializer.class)
    private Date birthday;
}
```

```
@Post("/")
public HttpResponse<Author> save(
    @Body Author author) {

    if(bookRepository.save(author)) {
      return HttpResponse.ok();
    } else {
      /* handle error */
    }
}
```

```
fetch("http://localhost:8080/
author/", {
    method: "POST",
    headers: new Headers({
        "Content-Type": "applicati
json"
    }),
    body: JSON.stringify({
        name: "Author's Name",
        birthday: "01/31/1985"
    })
})
```

JAVASCRIPT

https://www.baeldung.com/jackson-annotations

JACKSON: JSON RENDERING

```java
@JsonIgnoreProperties({"id", "version"})
public class Book {

    private Long id;
    private Long version;

    @JsonProperty("name")
    private String title;
    private Author author;
    private Integer pages;
    private List<String> tags;
}
```

```java
@Get("/{id}")
public Book show(Serializable id) {

    return bookRepository.get(id);
}
```

```json
{
  “name": "Title Here”,
  "author": {
    "name": "Author"
  },
  "pages": 150,
  "tags": [
    "tech",
    "bestseller"
  ]
}
```

JSON

https://www.baeldung.com/jackson-annotations

REST CONTROLLER

BOOKCONTROLLER.JAVA

```
@Controller("/book")
class BookController {

    @Post
    HttpResponse<BookDetails> save(@Valid @Body BookDetails bookDetails) { /* .. */}

    @Put
    HttpResponse<BookDetails> update(@Valid @Body BookDetails bookDetails) { /* .. */}

    @Delete("/{id}")
    HttpResponse delete(Serializable id) { /* .. */}

    @Get("{?max,offset}")
    @Transactional(readOnly = false)
    HttpResponse<List<Book>> list(@Nullable Integer max, @Nullable Integer offset) { /* .. */}

    @Get("/{id}")
    @Transactional(readOnly = true)
    HttpResponse<BookDetails> get(Serializable id) { /* .. */}

    HttpResponse<Integer> count() { /* .. */}

}
```

ENABLING CORS

- CORS support included in Micronaut
- Disabled by default
- Can specify allowed origins, methods, headers, max age, and more.

APPLICATION.YML

```yaml
micronaut:
    application:
        name: my-app
    server:
        cors:
            enabled: true
```

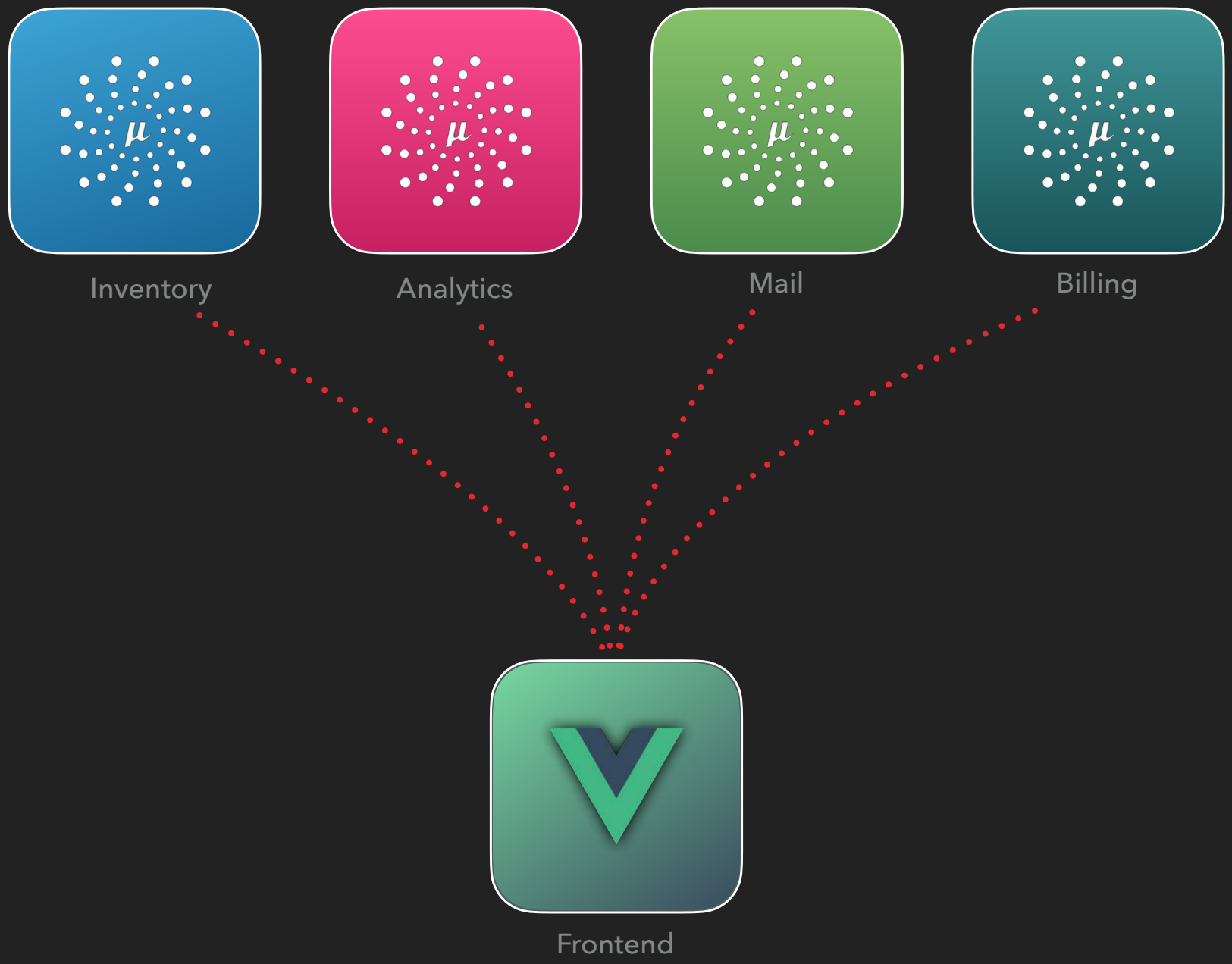
API GATEWAYS

Backend

REST

{JSON}

Frontend


Inventory
Analytics
Mail
Billing
Frontend

MICROSERVICES VS "TRADITIONAL" BACKEND

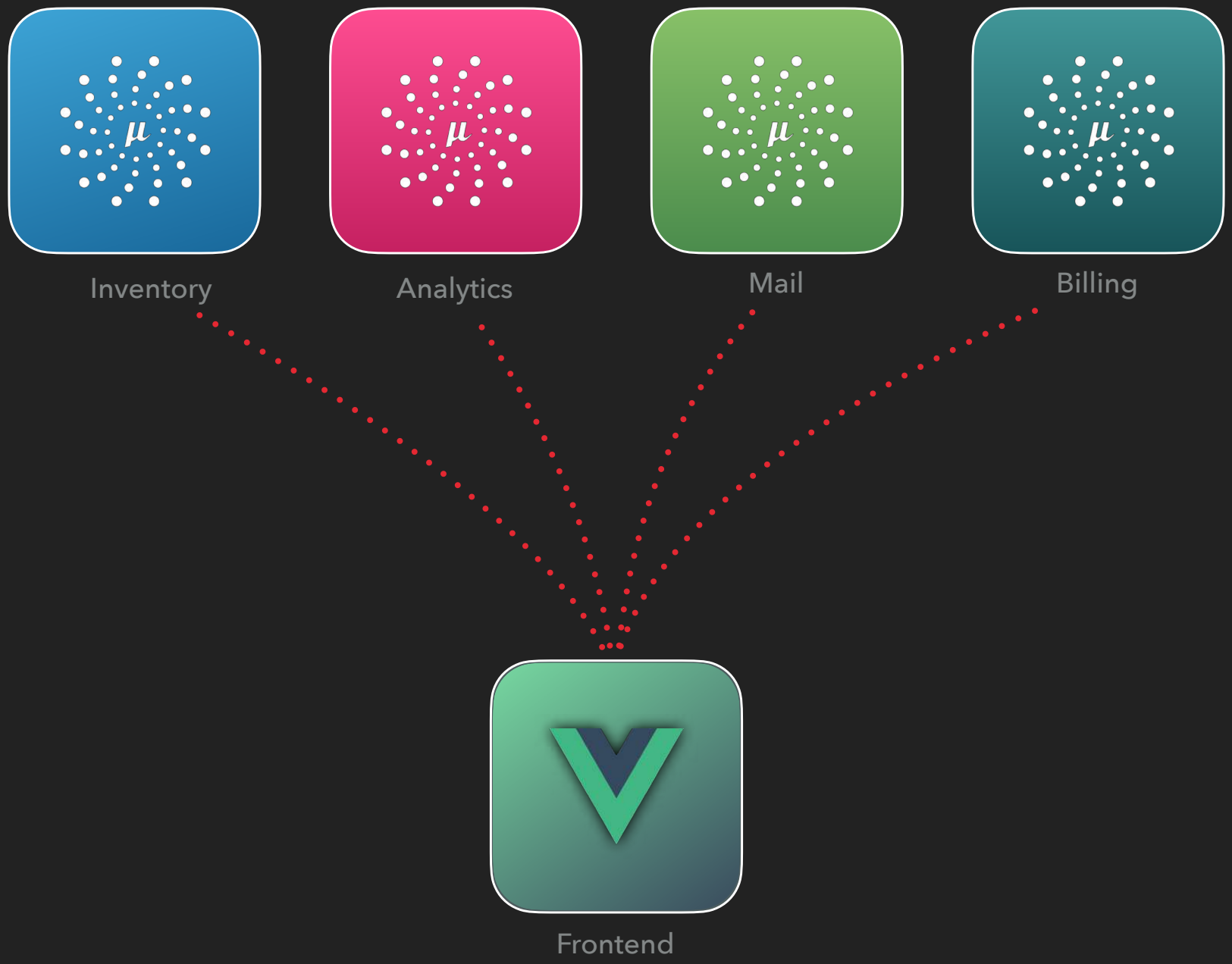
- Microservice Architectures implement service granularity
- Granularity offers many benefits, but complicates life for SPAs
- Services may be registered through service discovery (not known URLs)
- Not all clients (SPAs, mobile apps, traditional web apps) require the same data
- The frontend (SPA) shouldn't need to be "aware" of the topology of the backend system

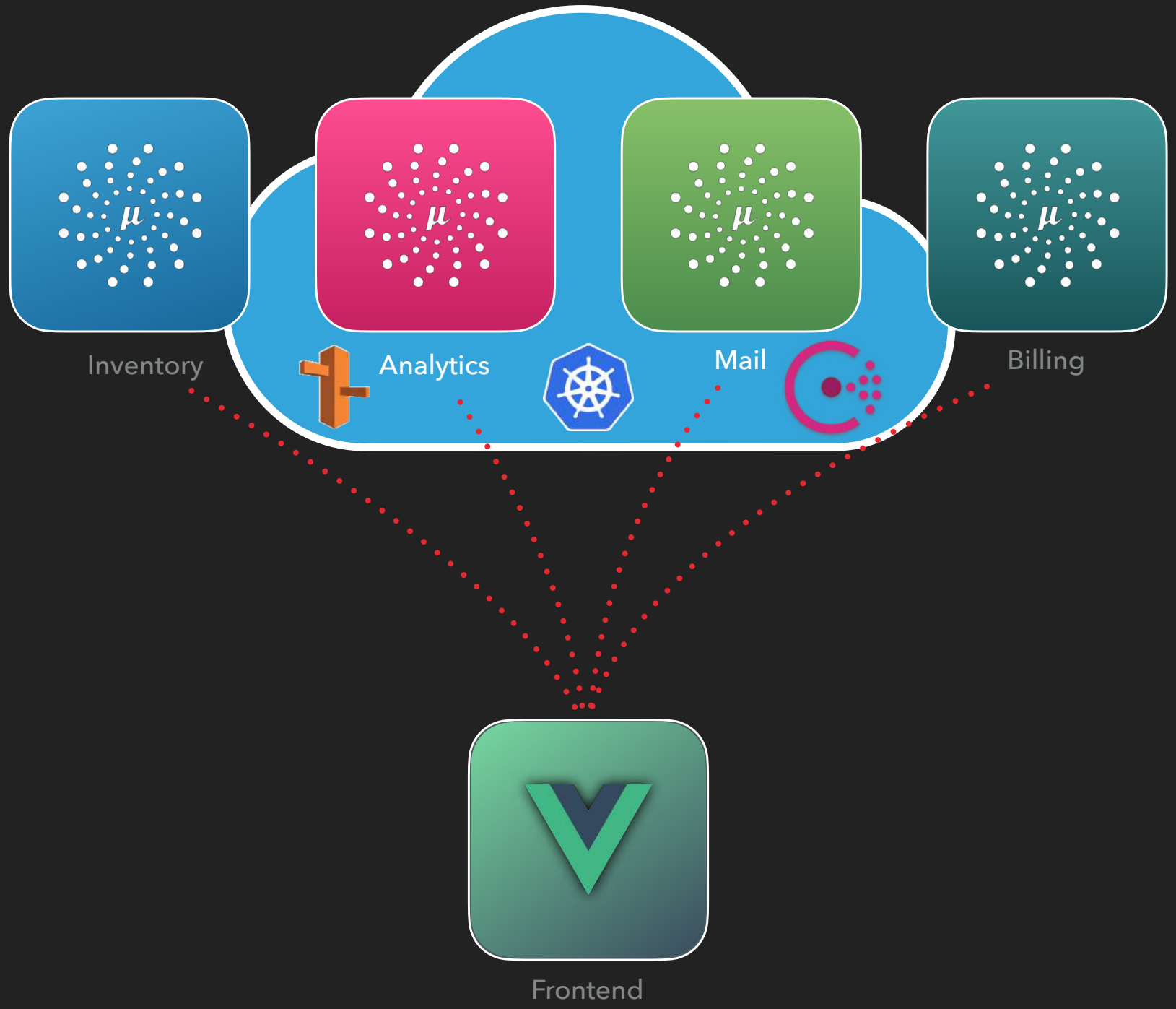
API GATEWAYS

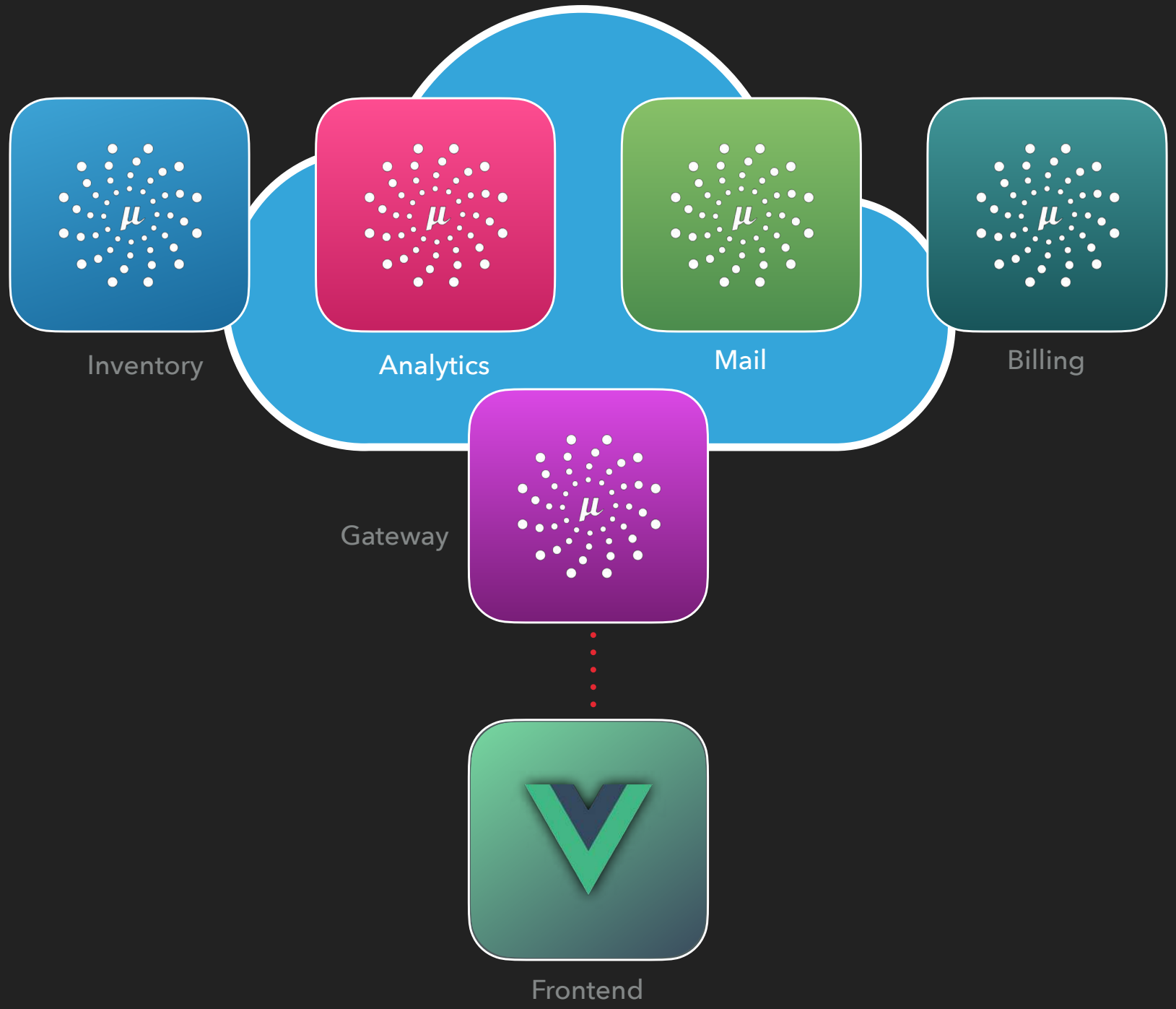
- Architectural pattern for microservice-based systems
- Expose a single client-facing API (for SPA, mobile, etc)
- Minimizing integration points - decoupling
- https://microservices.io/patterns/apigateway.html
- https://docs.microsoft.com/en-us/azure/architecture/microservices/design/gateway

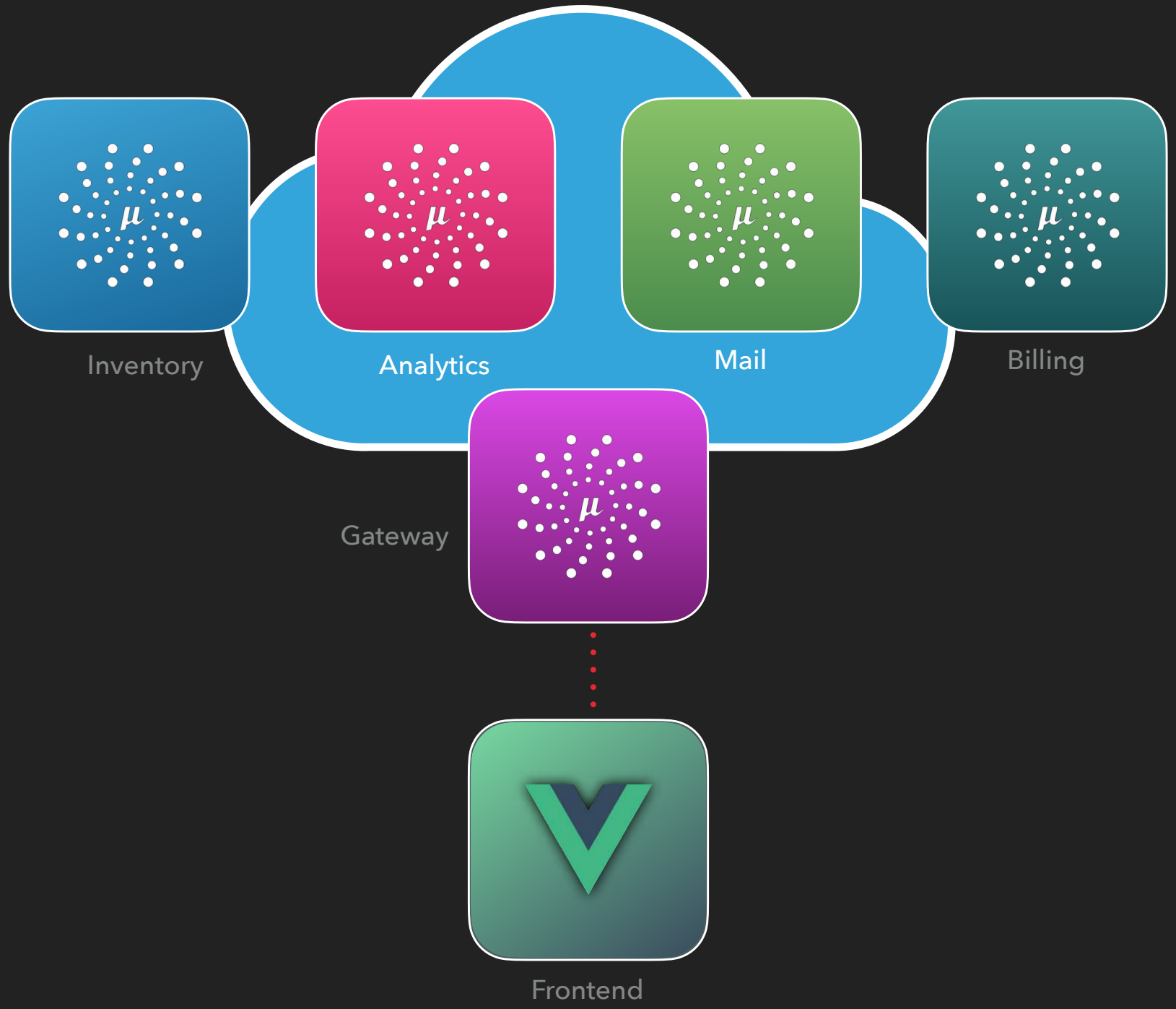
Backend

Frontend


Inventory
Analytics
Mail
Billing
Frontend


Inventory
Analytics
Mail
Billing
Frontend


Inventory
Analytics
Mail
Billing
Gateway
Frontend


Inventory
Analytics
Mail
Billing
Gateway
Frontend

API GATEWAYS

- Many features can be implemented at the API Gateway level:
  - Rate-limiting
  - Logging/tracing
  - Request aggregation
  - API Versioning
- Gateways should *not* be orchestrators!
- Open Source implementations (Netflix Zuul, Lyft Envoy, etc)
- Cloud providers (e.g, AWS) often supply their own API Gateway product
- Gateways can also be implemented as standalone services

BUILDING A GATEWAY WITH MICRONAUT

- Consistent APIs between controller (service) and client (gateway)
- Use of shared API libraries can simplify development
  - Shared API: `interface ProductAPI` - specifies API for product resource
  - Service: `ProductController implements ProductAPI` - specifies business logic
  - Gateway: `ProductClient extends ProductAPI` - consumes backend API on behalf of edge-clients
- Support for API versioning, tracing, load balancing, API docs, etc
- "Should I Make My Own API Gateway?" - https://www.youtube.com/watch?v=YO6Sg4yaqC0

API DOCUMENTATION

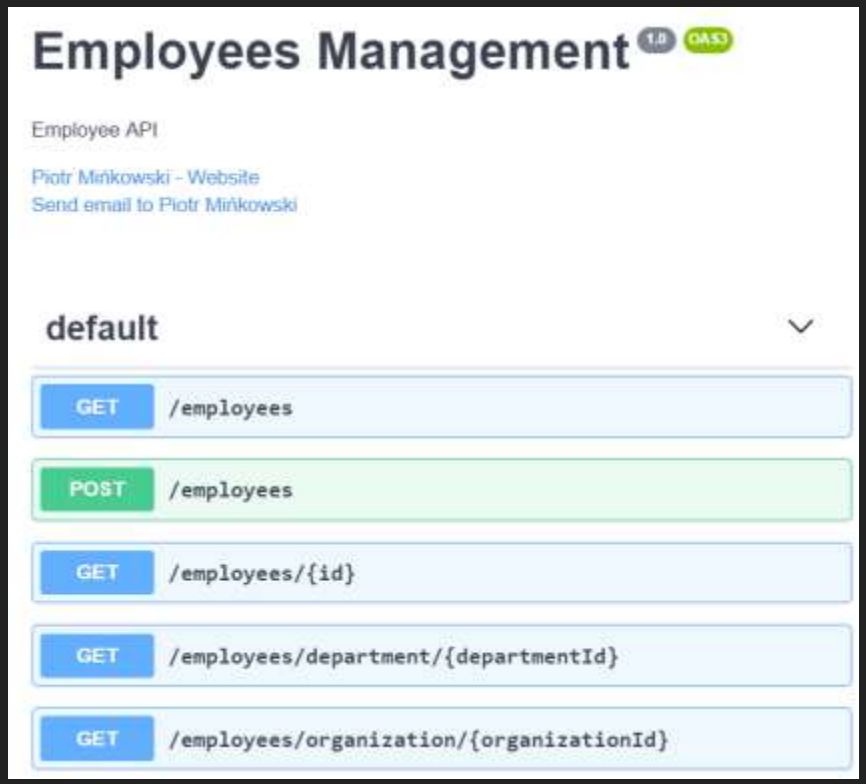
- Micronaut can generate OpenAPI (Swagger) YAML definitions at compilation time
- Standard Micronaut annotations (`@Controller`, `@Get`, `@Consumes`, etc) and method return types (POJOs) will be analyzed and corresponding Swagger YML written to the file
- Standard Swagger annotations can be used to customize/override the generated YAML
- Micronaut can handle merging of OpenAPI schemas from multiple modules (e.g., when using Micronaut Security)

API DOCUMENTATION

Configuration to expose Swagger YAML over the server:

src/main/resources/application.yml

```
micronaut:
  router:
    static-resources:
      swagger:
        paths: classpath:META-INF/swagger
        mapping: /swagger/**
```



Swagger

https://micronaut-projects.github.io/micronaut-openapi/latest/guide/index.html

MICRONAUT API VERSIONING

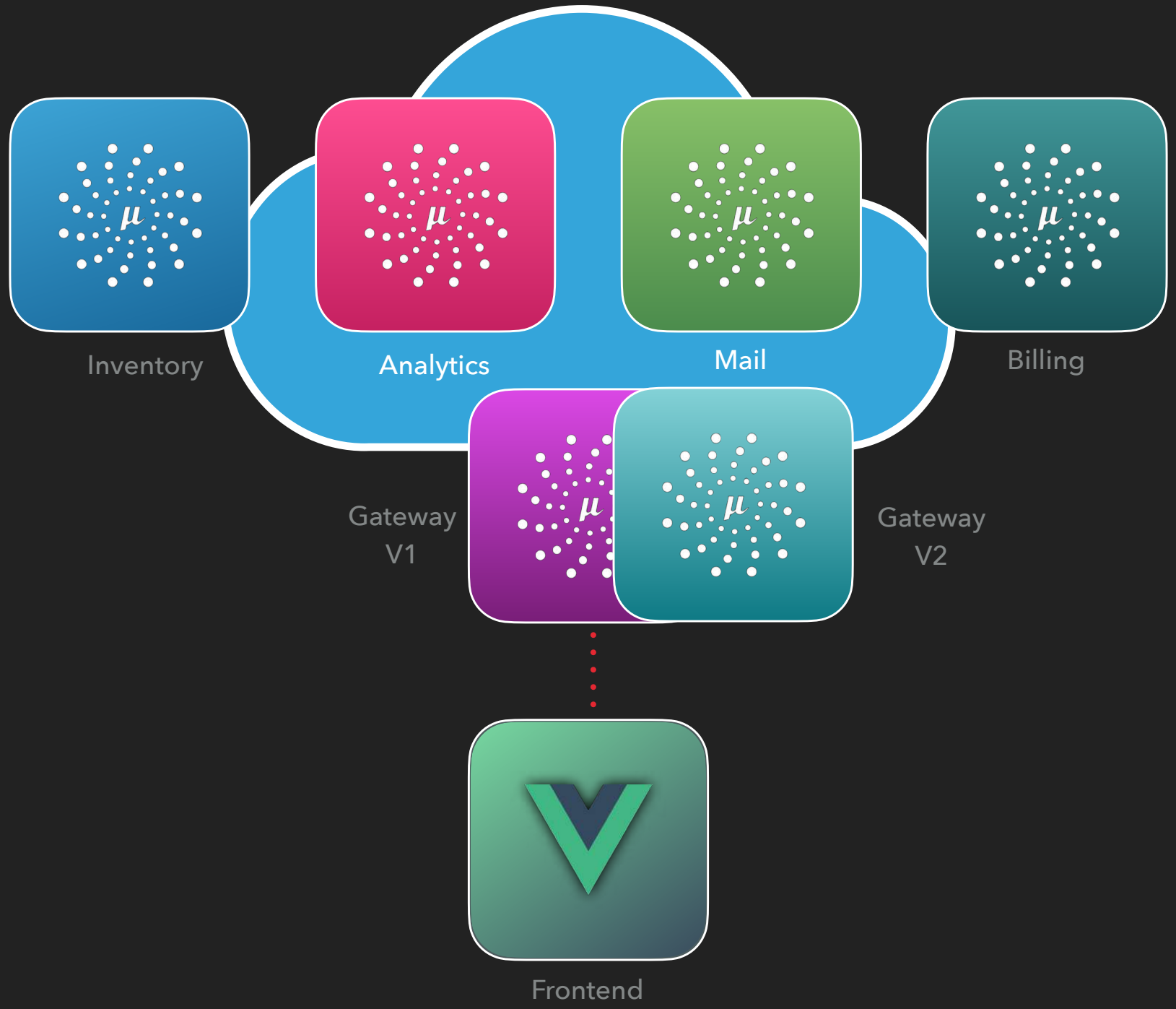
- Version by URL: `@Get("/v1/user/profile")`
  - Using config property:

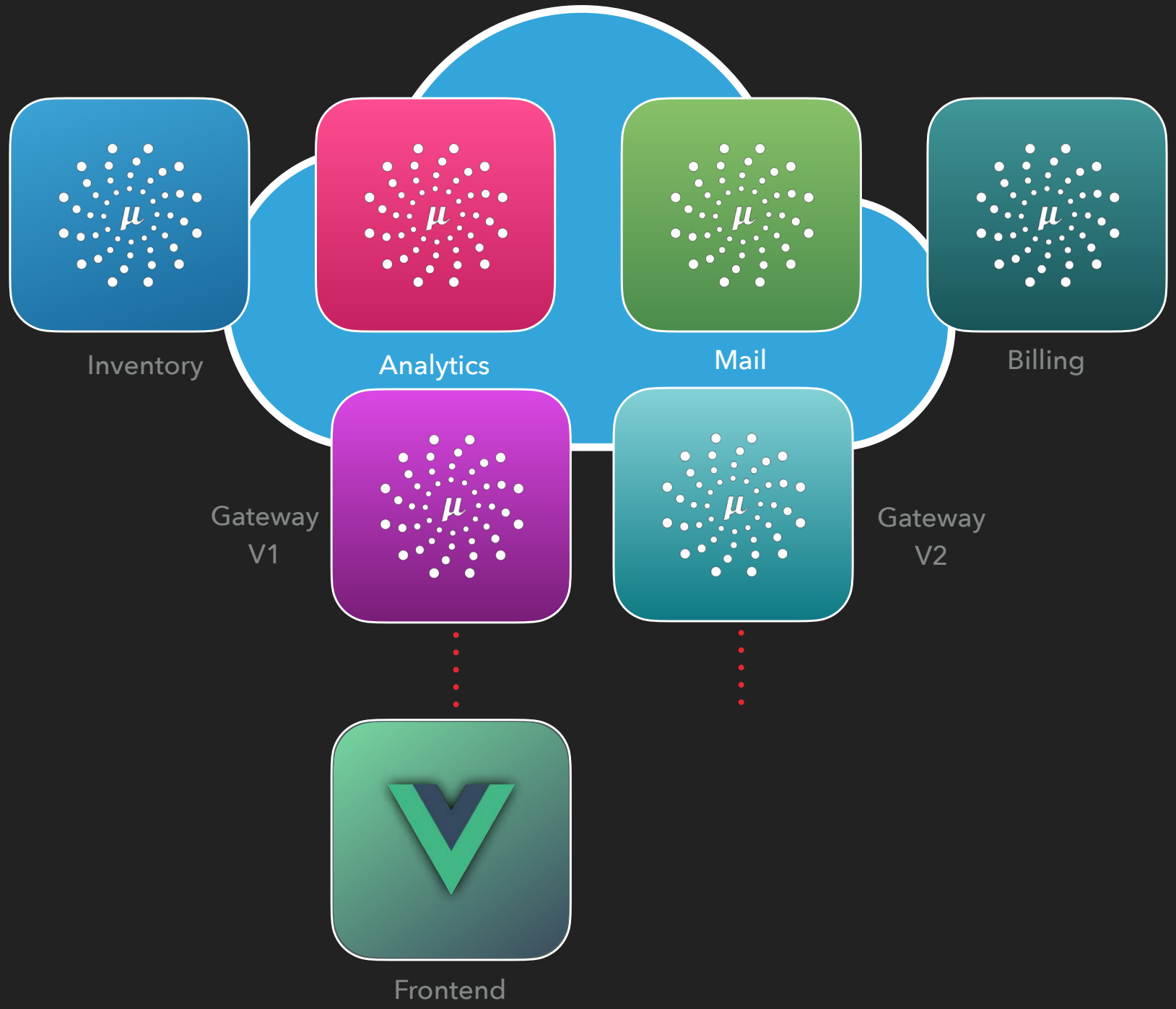
```
@Value("${core.api.version}")
String version

@Get("/${version}/user/profile")
```

APPLICATION.YML

```yaml
core:
  api:
    version: v1
```

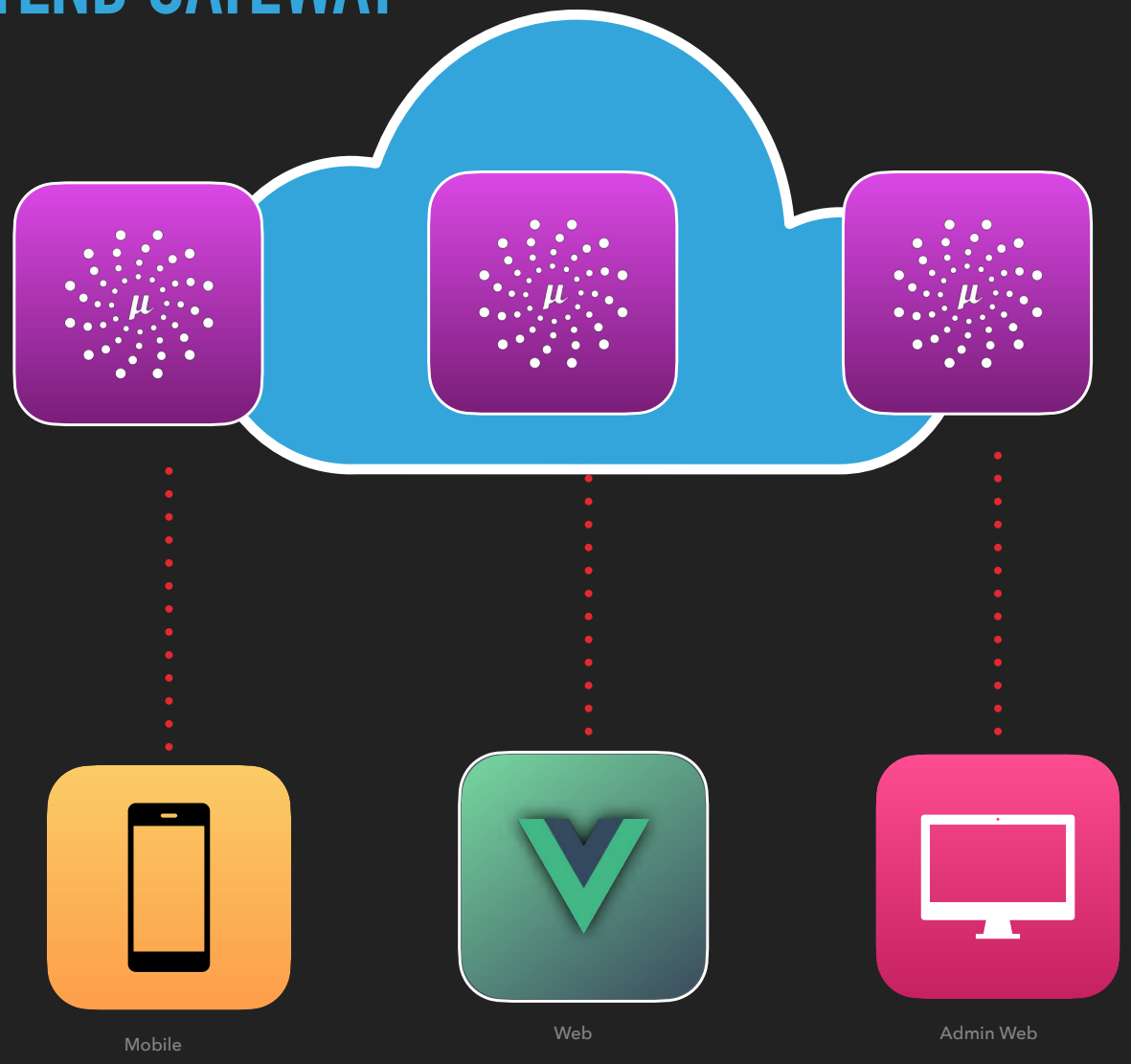
- Client-facing versioning can be separate from versioning within the microservice architecture


Inventory
Analytics
Mail
Billing
Gateway
V1
Gateway
V2
Frontend

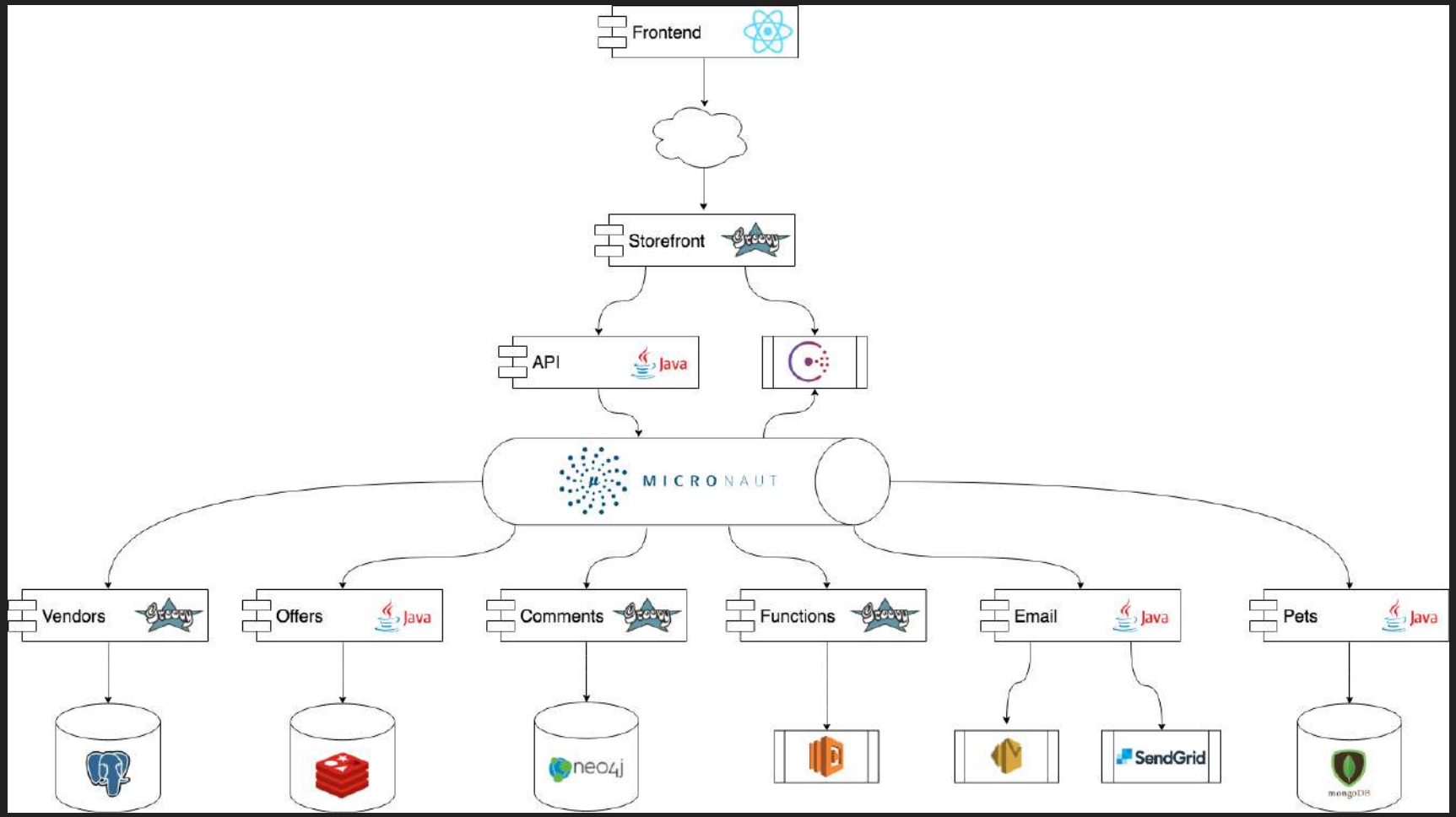

Inventory
Analytics
Mail
Billing
Gateway
V1
Gateway
V2
Frontend

BACKEND PER FRONTEND GATEWAY

- Partition your API
- Support different client needs (web vs mobile etc)



MICRONAUT PETSTORE

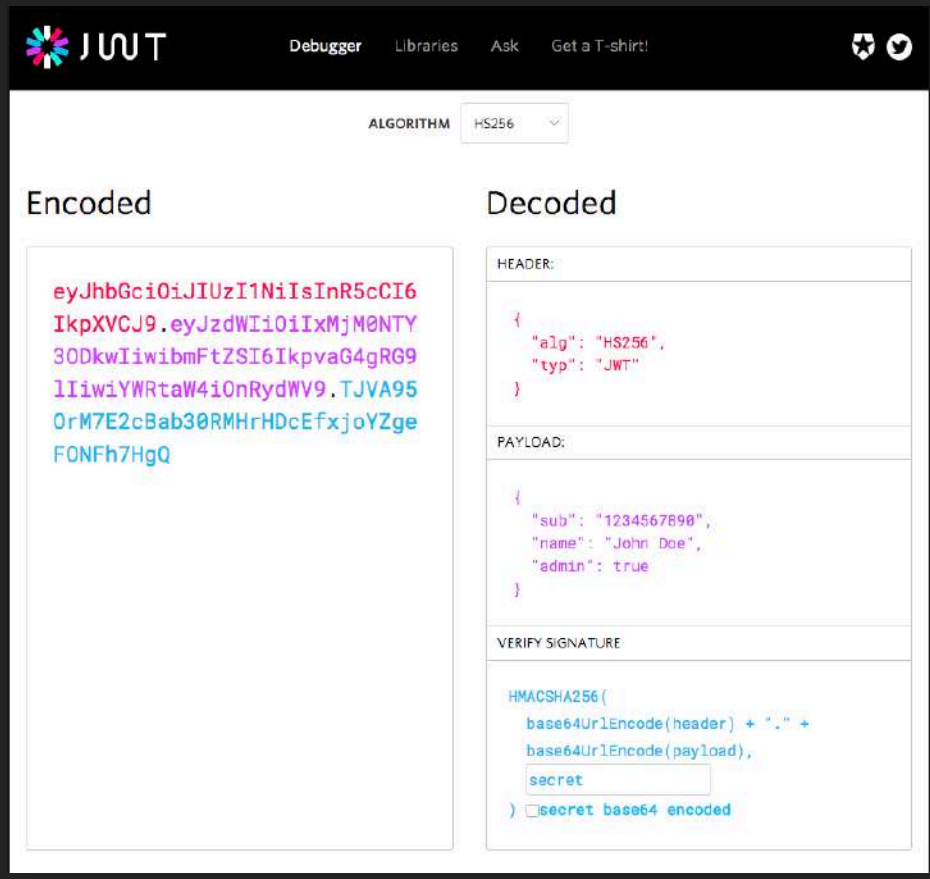


https://github.com/micronaut-projects/micronaut-examples/tree/master/petstore

SECURITY WITH JWT

JWT: JSON WEB TOKEN

- Open standard for representing claims securely between two parties
- Tokens can be signed with either a secret or public/private key
- Standard approach for stateless authentication
- Ideal for transmitting authentication & authorization data between microservices and single-page-apps



MICRONAUT SECURITY

- Core Micronaut Library - supports JWT, OAuth 2.0
- Annotation-based API & config-based URL mappings
- Support for **token propagation**
- Supports RFC 6750 Bearer Token
- JWTs can be read from cookie

APPLICATION.YML

```
micronaut:
  security:
    enabled: true
    token:
      jwt:
        enabled: true
        signatures:
          secret:
            generator:
              secret: changeMe
```

BUILD.GRADLE

```
dependencies {
   compile "io.micronaut:micronaut-security-jwt"
}
```

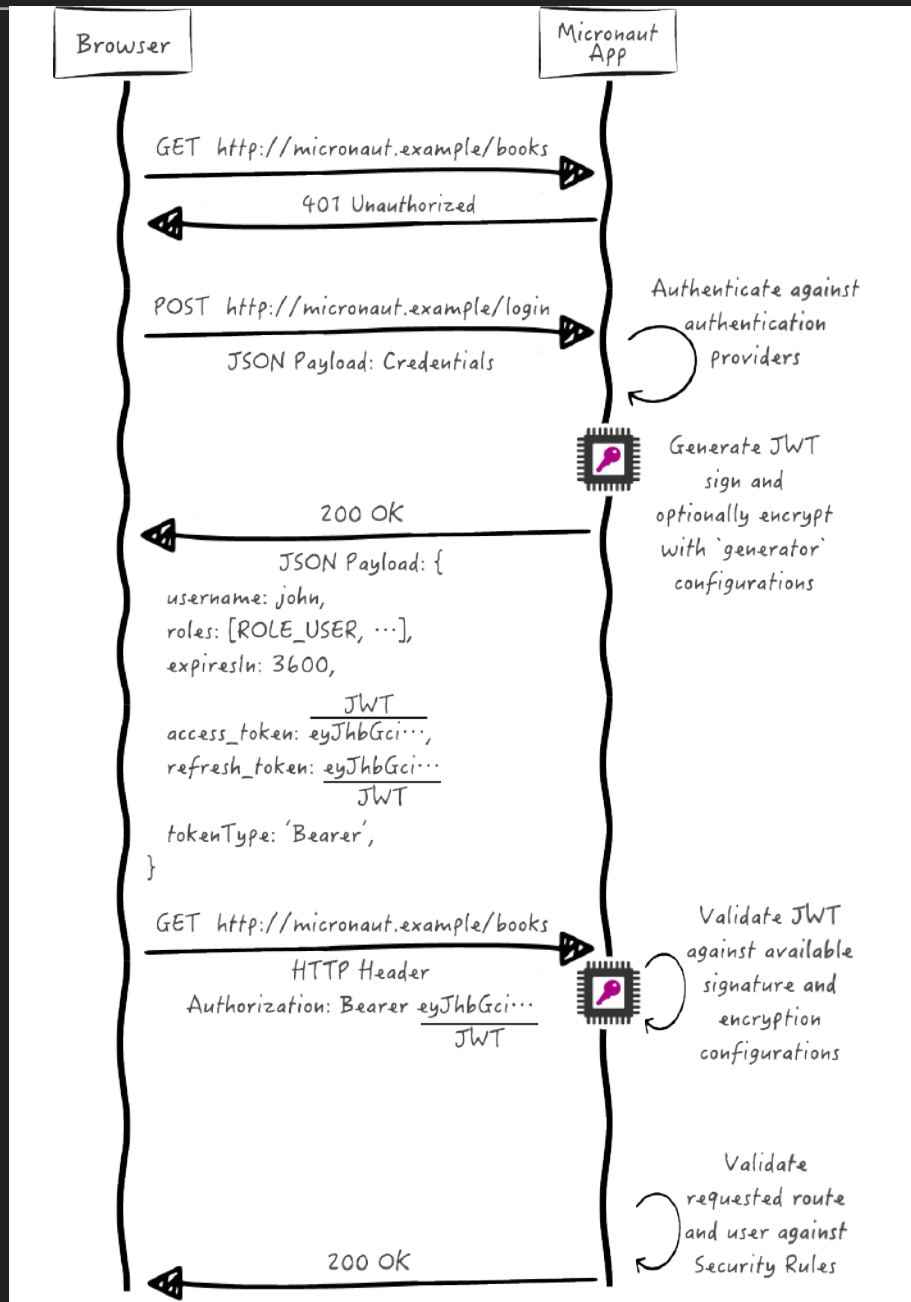
@SECURED ANNOTATION

- @Secured annotation applied to controllers and methods
- All routes blocked by default
- Can require authentication and/or authorization (role-based)
- Alternative: JSR-250 security annotations are also supported: `@PermitAll`, `@RolesAllowed`, `@DenyAll`

```
import java.security.Principal;

@Secured("isAuthenticated()")
@Controller("/")
public class HomeController {

    @Get("/")
    String index(Principal principal) {
        return principal.getName();
    }

    @Secured({"ROLE_ADMIN", "ROLE_X"})
    @Get("/classified")
    String classified() {
        return /* REDACTED */;
    }
}
```

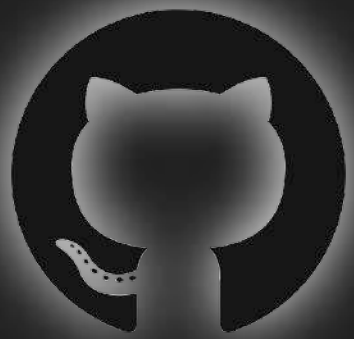
MICRONAUT JWT SECURITY

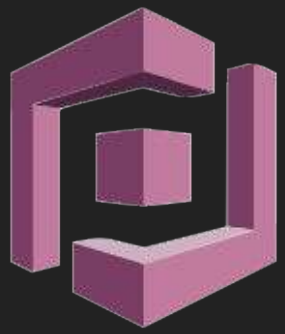
- Unauthorized request is made to API
- Responds with 401
- Client POSTs to login endpoint
- Server responds with JWT
- Client includes access token in the Authorization header for subsequent requests
- Server validates the incoming token
- If authorized, server responds with resource



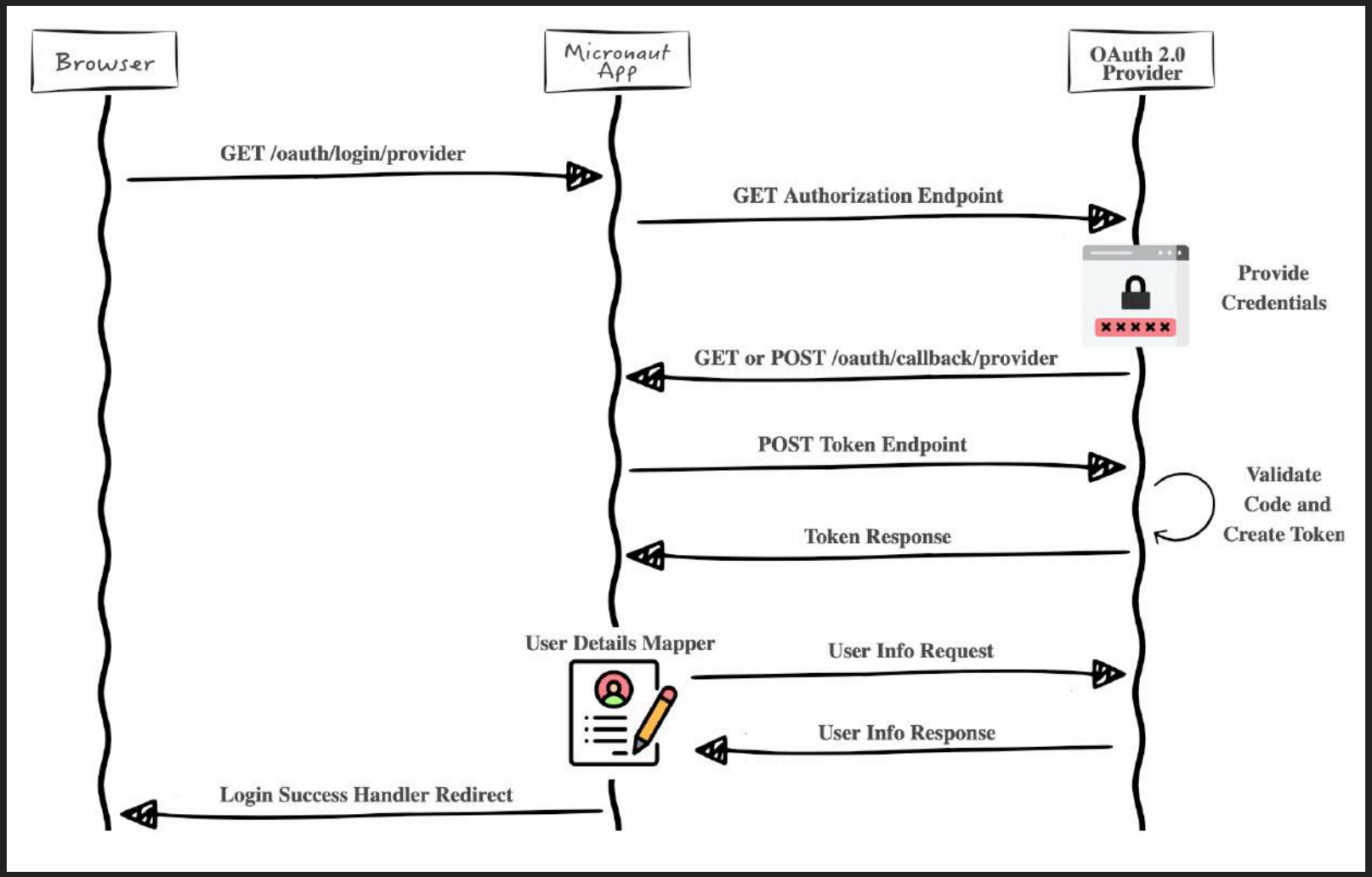
OAUTH 2.0

- Delegate authentication to a third-party provider (or custom provider)
- Requires a mapping between the provider's auth and user identity and authorization (within your application)
- Typically configured via a client ID/secret pair and a callback URL





OAUTH 2.0


Browser
Micronaut App
OAuth 2.0 Provider
GET /oauth/login/provider
GET Authorization Endpoint
Provide Credentials
GET or POST /oauth/callback/provider
POST Token Endpoint
Validate Code and Create Token
Token Response
User Details Mapper
User Info Request
User Info Response
Login Success Handler Redirect

MICRONAUT SECURITY & OAUTH GUIDES

GUIDES FILTERED BY #SECURITY

Micronaut Basic Auth
JAVA KOTLIN GROOVY

Session based authentication
JAVA GROOVY KOTLIN

Micronaut JWT Authentication
JAVA GROOVY KOTLIN

Micronaut JWT authentication via Cookies
JAVA GROOVY KOTLIN

LDAP and Database authentication providers

https://guides.micronaut.io/tags/security.html

GUIDES FILTERED BY #OAUTH2

Secure a Micronaut app with Okta

Secure a Micronaut app with Google

Secure a Micronaut app with Cognito

Secure a Micronaut app with Github

Secure a Micronaut app with Github

https://guides.micronaut.io/tags/oauth2.html

TOKEN PROPAGATION

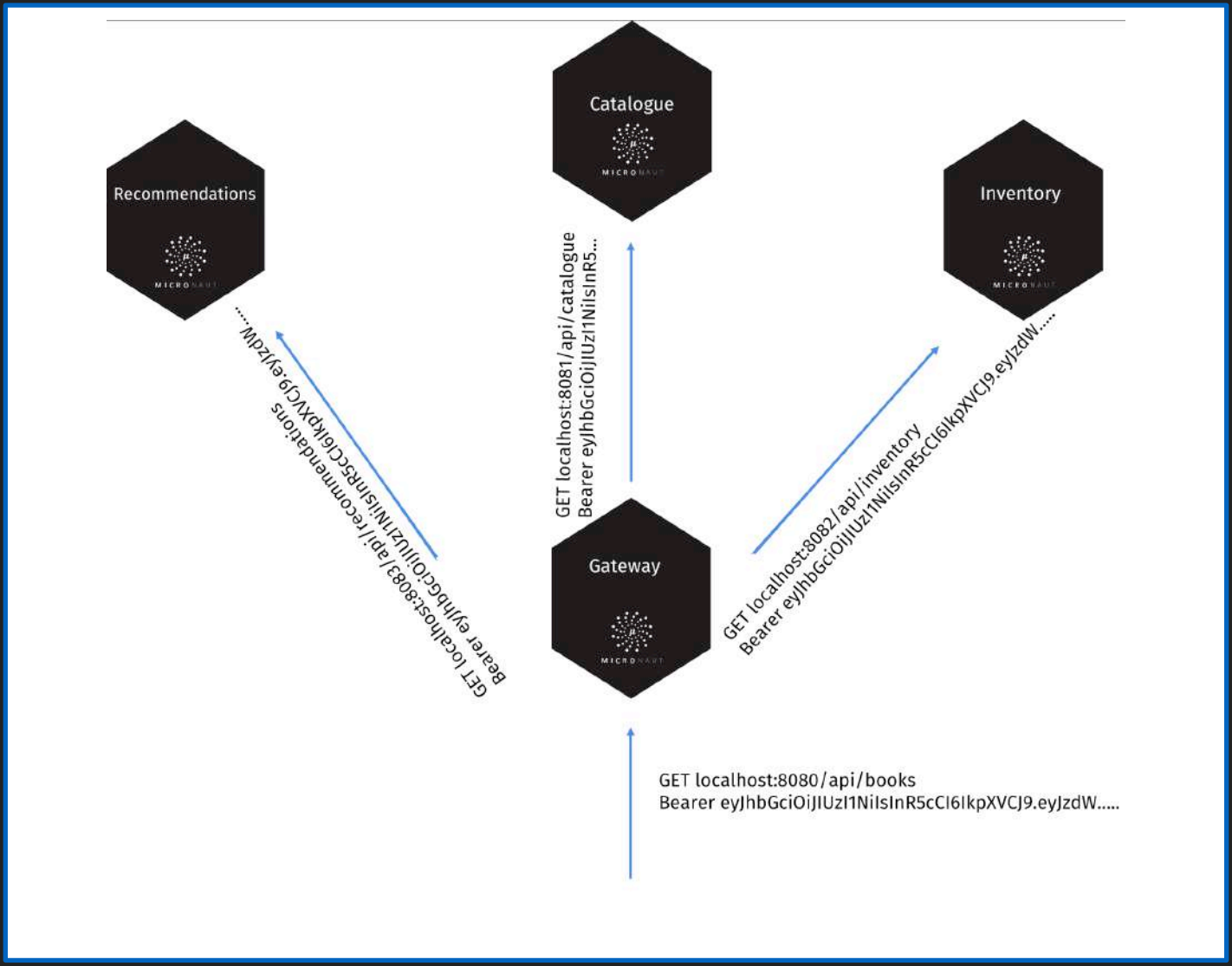
TOKEN PROPAGATION

- Micronaut can embed an access token within the request
- Token can be stored as a cookie, or within an HTTP Header
- Services to which tokens should be propagated can be specified via config
- Allows each service to enforce authentication/authorization

APPLICATION.YML

```yaml
micronaut:
  security:
    enabled: true
    token:
      jwt:
        enabled: true
      writer:
        header:
          enabled: true
      propagation:
        enabled: true
        service-id-regex: "inventory"
```

SINGLE PAGE APPS FOR A MICROSERVICE ARCHITECTURE



https://guides.micronaut.io/micronaut-token-propagation

MULTITENANCY

OCI

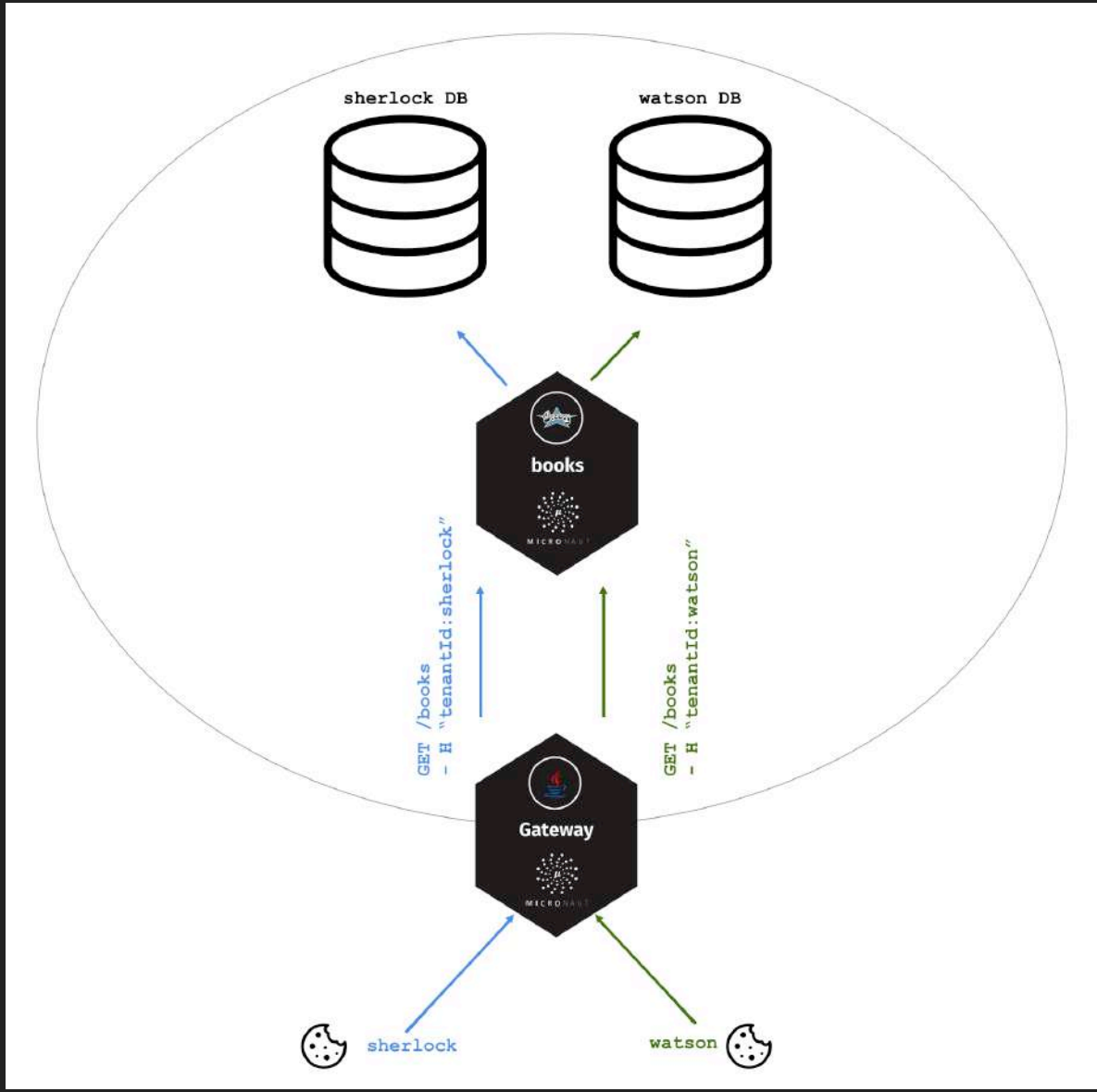
MULTITENANCY

- An approach for partitioning user data within the application state (e.g, database)
- Micronaut supports tenant propagation across services
- Can *read* tenant from HTTP header, Cookie, or Subdomain (acme.mysite.com)
- Can *write* tenant to HTTP header or Cookie

APPLICATION.YML

```yaml
micronaut:
  multitenancy:
    propagation:
      enabled: true
      service-id-regex: 'inventory'
    tenantresolver:
      httpheader:
        enabled: true
    tenantwriter:
      httpheader:
        enabled: true
```

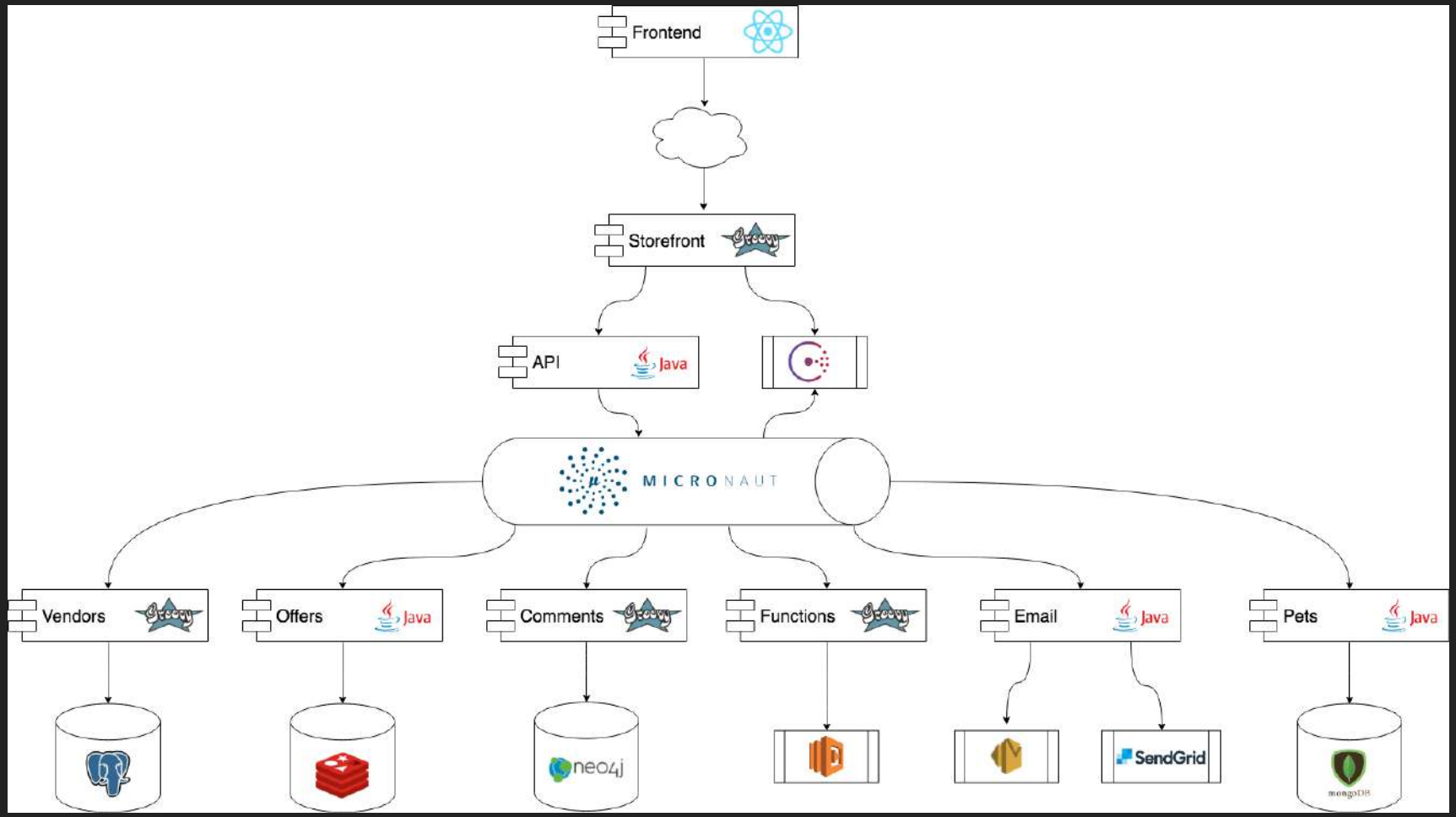
```
@Get("/")
List<ProductDetails> list(@Header tenantId) {

}
```



https://guides.micronaut.io/micronaut-multitenancy-propagation

DEMO

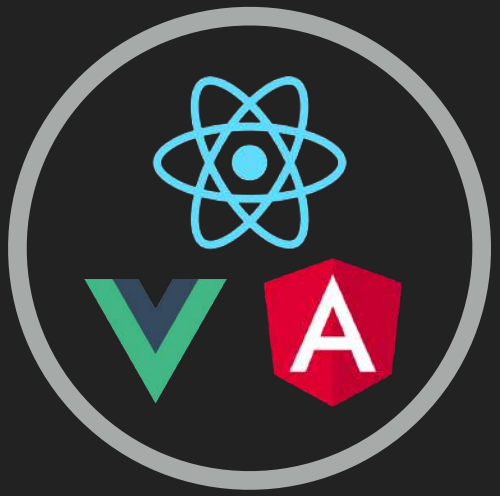
MICRONAUT PETSTORE



https://github.com/micronaut-projects/micronaut-examples/tree/master/petstore

SUMMARY

- Micronaut is a powerful microservice solution
- SPAs (and other clients) must be considered in architecture design
- API Gateways are a powerful approach
- In lieu of third-party solutions, Micronaut makes an excellent choice for building custom gateways if required
- Token propagation (for security, multitenancy) simplifies interservice communication for SPAs and other edge-clients



LEARN MORE ABOUT OCI EVENTS AND TRAINING

OBJECT COMPUTING
HOME TO GRAILS & MICRONA

Events:

- objectcomputing.com/events

Training:

- objectcomputing.com/training
- grailstraining.com
- micronauttraining.com

Or email 2GM@objectcomputing.com to schedule a custom training program for your team online, on site, or in our state-of-the-art, Midwest training lab.

OBJECT COMPUTING

HOME TO GRAILS & MICRONAUT

CONNECT WITH US

1+ (314) 579-0066

@objectcomputing

objectcomputing.com

OBJECT COMPUTING
HOME TO GRAILS & MICRONAUT

WEBINAR

Single Page Apps for a Microservices Architecture

Presented by Zachary Klein

MICRONAUT