REST-API in AWS with Lambda.

An early approach on how to build a serverless REST-API.

Project

1. Pivoted from a terminated project (alas)
2. Product owner who has opinions-that-they-can-state-clearly about:
    a. Non-functional requirements
    b. Functional requirements
3. Actual users you can talk to
4. Functional architect who can formalise functional requirements as features
5. Self-organizing team (one-size does not fit all)
6. Iterative development

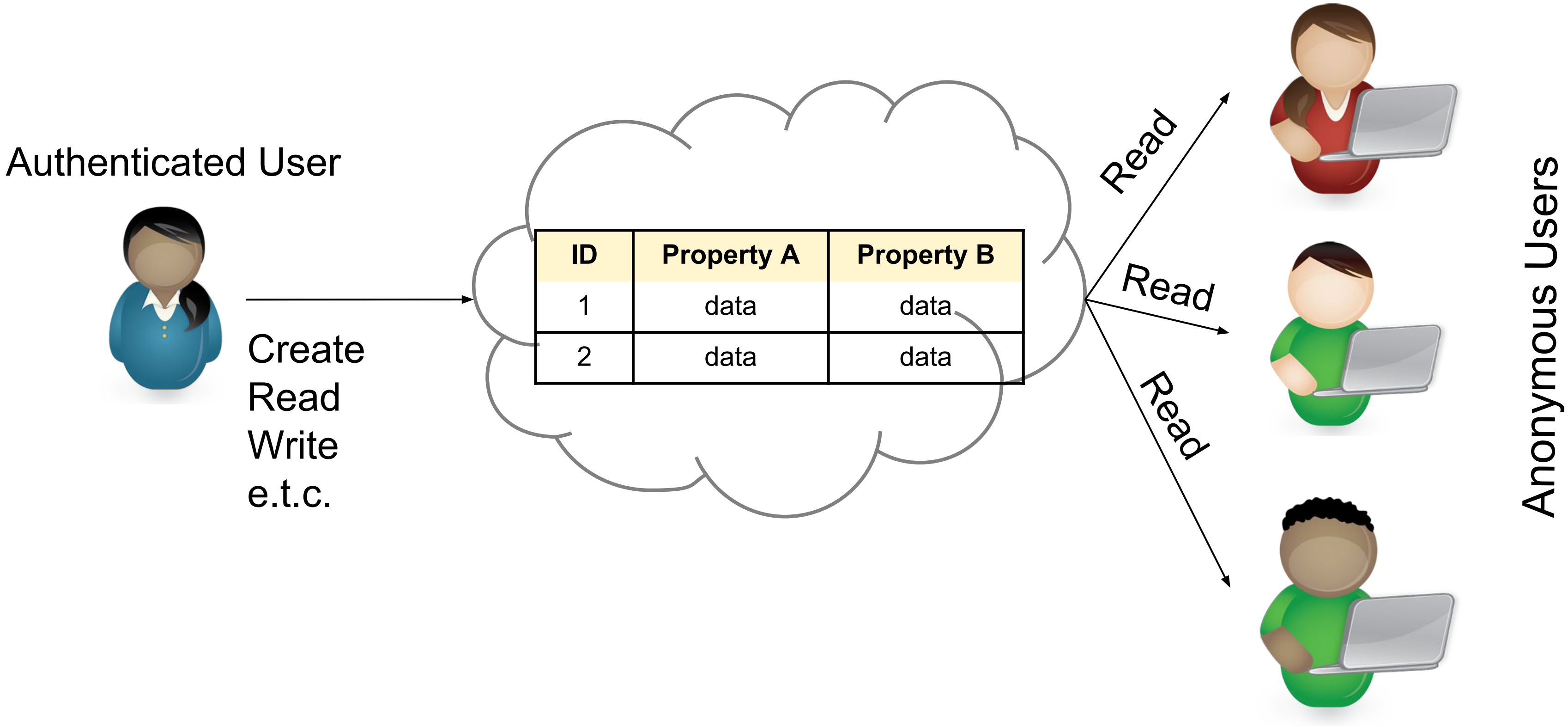
Non-functional requirements

1. That the service is outside current library system
2. In public cloud (and not SafeSpring)
3. Managed services / FaaS
4. Semantic web
5. Gradle

Customer needs

1. Clients need to create their own Entity-registry with user-specified data-schema.
2. Clients need to be able to
   a. create new registries.
   b. write in their registries.
   c. read from their registries.
3. Clients wish that anonymous users will be able to read data from their registry.

Customer needs



Authenticated User

Create
Read
Write
e.t.c.

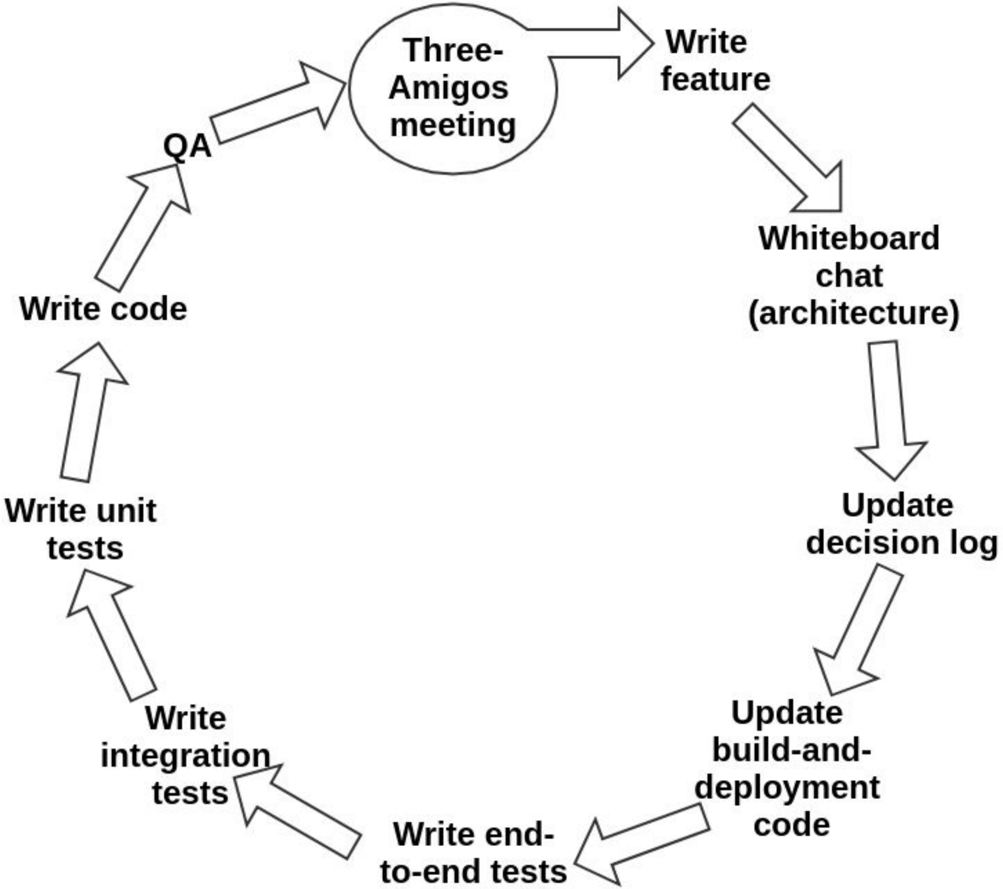
ID	Property A	Property B
1	data	data
2	data	data

Read

Read

Read

Anonymous Users



Three-Amigos meeting

Write feature

Whiteboard chat (architecture)

Update decision log

Update build-and-deployment code

Write end-to-end tests

Write integration tests

Write unit tests

Write code

QA

Gherkin

```gherkin
Feature: Admin user features

 Scenario: An API admin user provides a valid API key
  Given that an API admin user has a valid API key for API administration
  When they submit the API key
  Then they can access the administration APIs

 Scenario: An registry admin user adds a single entity to a registry
 Given that the registry admin user has a valid API key for registry administration
 And that there is an existing entity registry with a schema
 When the registry admin user submits the API key with a request to create a new entity
          with properly formatted data
 Then the entity is created
```

Gherkin

```
Scenario: An anonymous user views an entity specifying an RDF serialization
  Given that there is an existing entity registry with a schema
  And that there is an entity in the registry
  When the anonymous user requests the entity specifying an Accept header with value:
    | application/ld+json   |
    | application/n-triples |
    | application/rdf+xml   |
    | application/turtle    |
    | application/json      |
    | application/rdf       |
Then anonymous user can view the data in the given serialization
```

How do we use Gherkin?

- A **formal** specification of the desired features
- A **single point of truth** for what has been agreed to be produced
- A **log** (because it is code in version control) of the evolution of customer needs
- A **verification** that the product is ready (acceptance test)

Automated acceptance testing with Gherkin & Cypress.io glue

```
let credentials = "";
let authenticationUrl = "https://www.unit.no"; // authentication service here
let authenticated = 'not authenticated';


given('that there is an API admin user with valid credentials', () => {
        credentials = "API admin user credentials";
})


when('they provide these credentials', () => {
        cy.request(authenticationUrl, credentials)
                .then((response) => { // check if authenticated
                        authenticated = 'authenticated';
                        cy.wrap(authenticated).as('authenticated')
                })
})


then('they are authenticated and receive a valid authentication token', () => {
        cy.get('@authenticated').should('equal', 'authenticated')
})
```

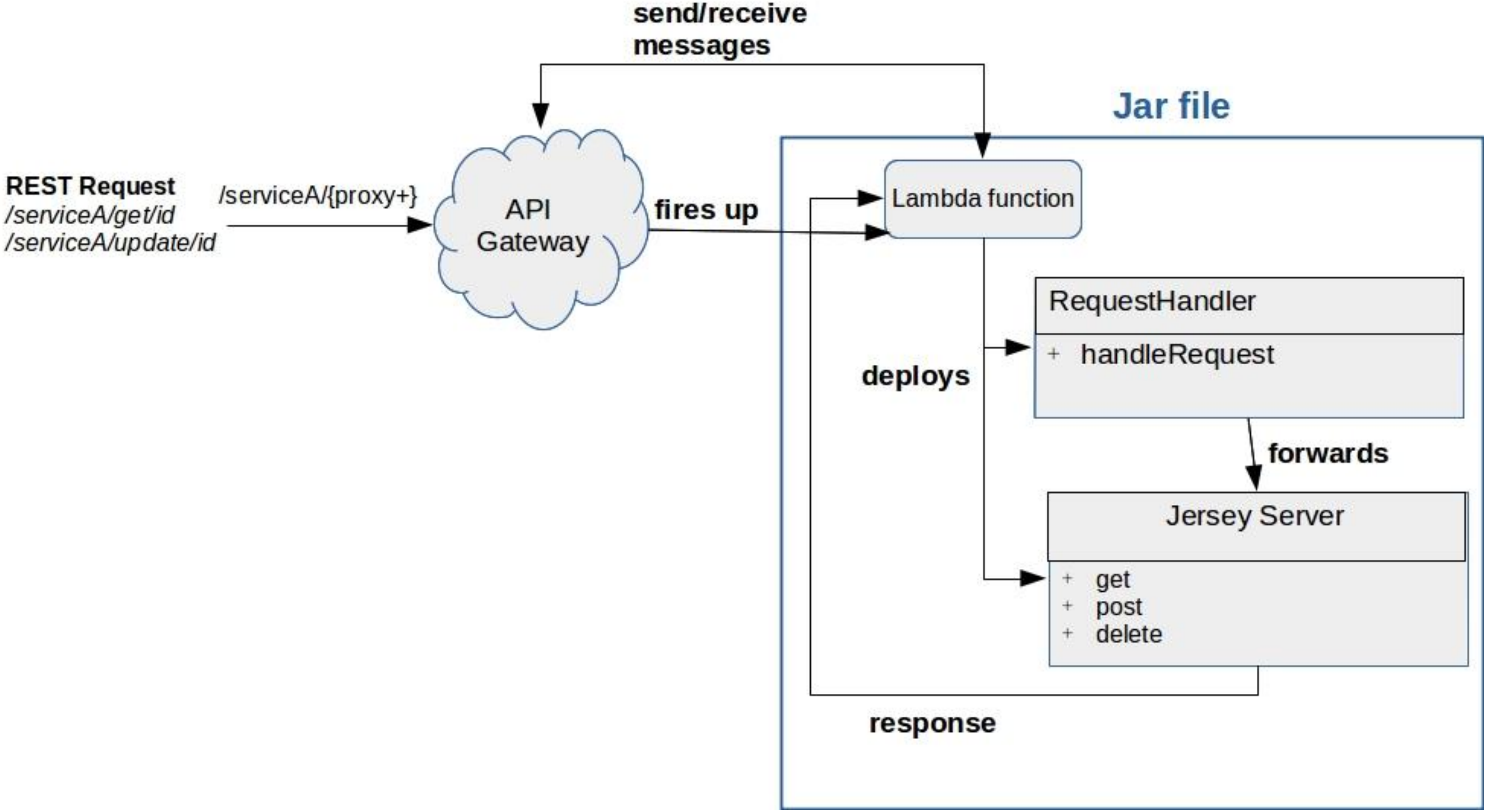
Advantages of using Gherkin

1. When all tests are green, we have a deliverable product
2. When all tests are green, we know we have satisfied all the client’s / product owner’s specification requirements
3. Our system is being tested for all the specified features every time we deploy our code (end-to-end testing)
4. The deployment fails if one feature test fails

An AWS Serverless Application

- Lambda
- API Gateway
- DynamoDB
- CodePipeline

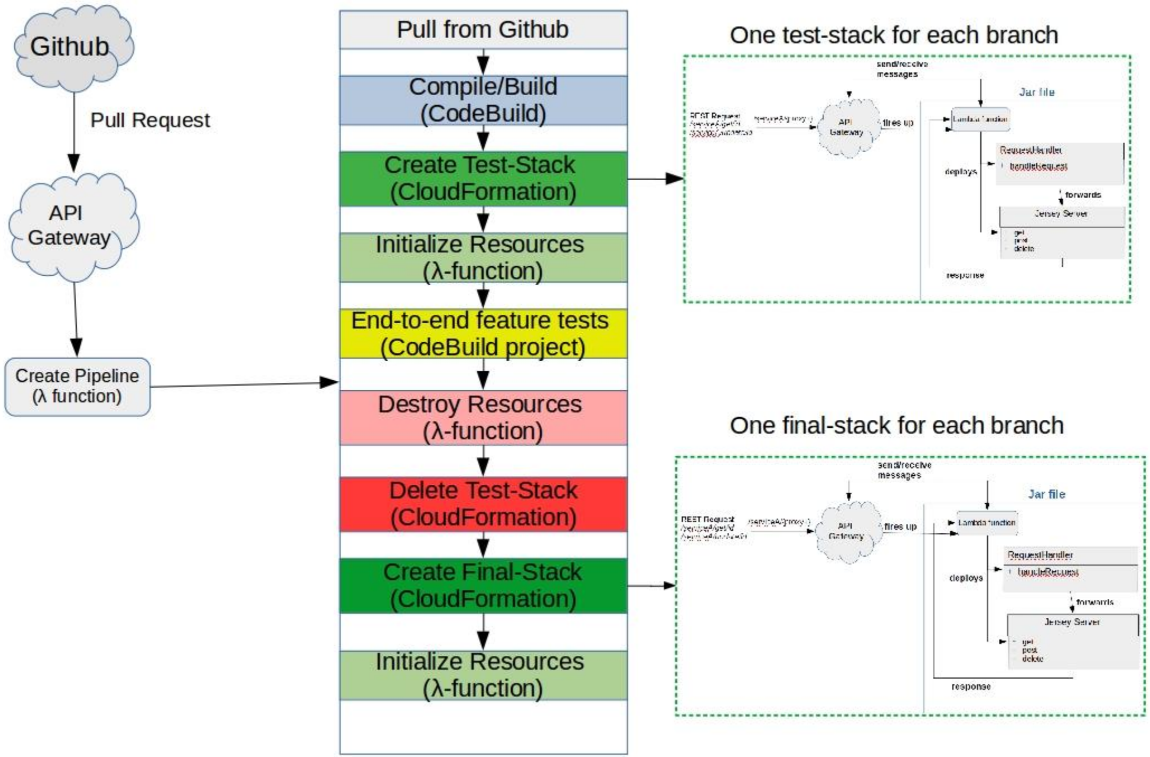
System architecture



Challenges deploying code in AWS

- No “Multi-branch Pipelines” feature in AWS CodePipeline
- CodePipeline is immature
  - Integration expectations between Github and AWS are not met
- No previous experience with deploying in the Cloud

Dynamic Pipelines



Github

Pull Request

API Gateway

Create Pipeline (λ function)

Pull from Github

Compile/Build (CodeBuild)

Create Test-Stack (CloudFormation)

Initialize Resources (λ-function)

End-to-end feature tests (CodeBuild project)

Destroy Resources (λ-function)

Delete Test-Stack (CloudFormation)

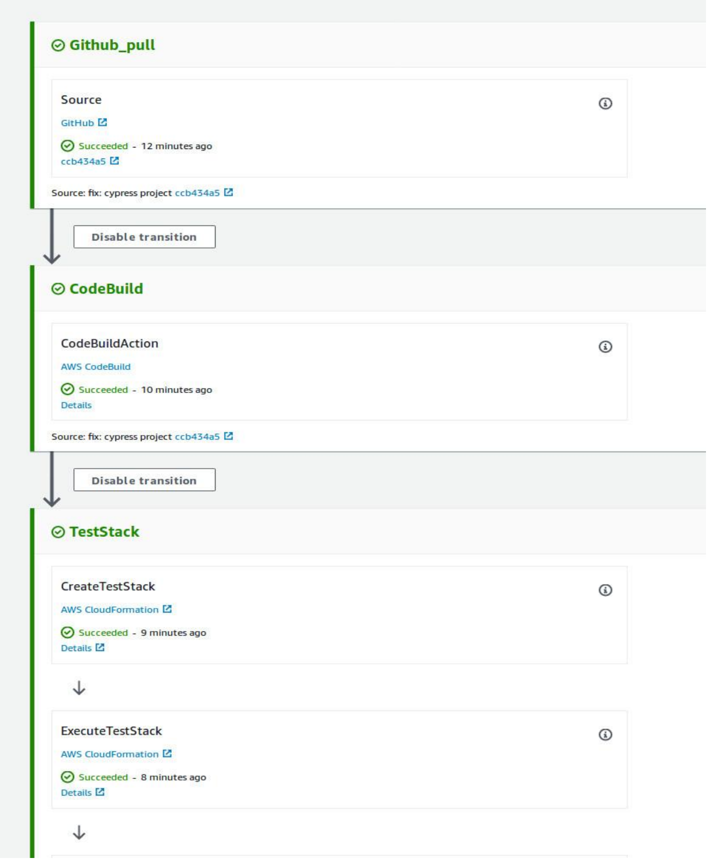
Create Final-Stack (CloudFormation)

Initialize Resources (λ-function)

One test-stack for each branch

One final-stack for each branch

CodePipeline Example



✓ Github_pull

Source
GitHub
✓ Succeeded - 12 minutes ago
ccb434a5

Source: fix: cypress project ccb434a5

✓ CodeBuild

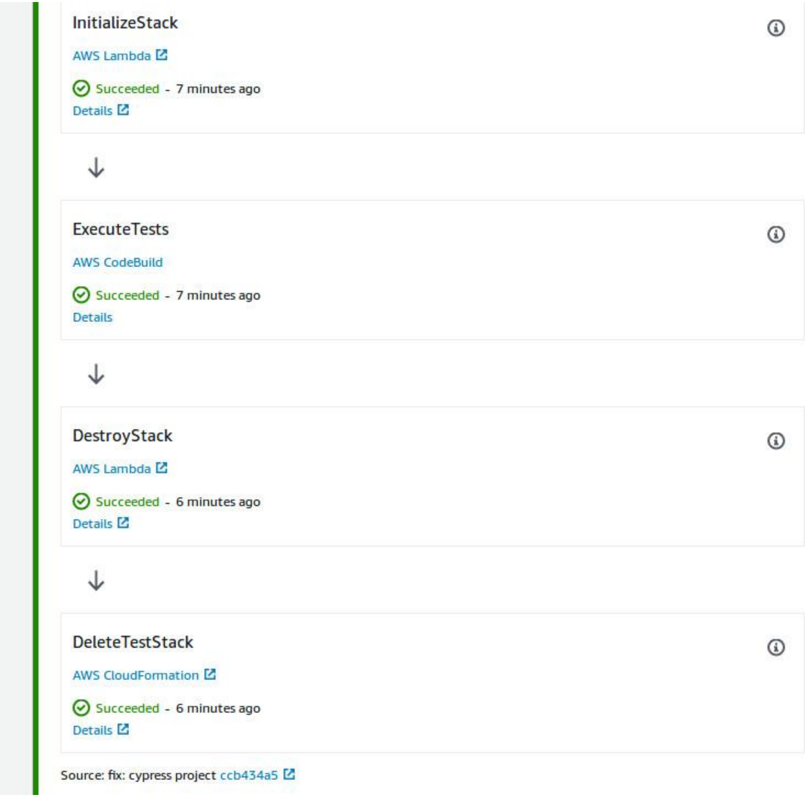
CodeBuildAction
AWS CodeBuild
✓ Succeeded - 10 minutes ago
Details

Source: fix: cypress project ccb434a5

✓ TestStack

CreateTestStack
AWS CloudFormation
✓ Succeeded - 9 minutes ago
Details

ExecuteTestStack
AWS CloudFormation
✓ Succeeded - 8 minutes ago
Details



InitializeStack
AWS Lambda
✓ Succeeded - 7 minutes ago
Details

ExecuteTests
AWS CodeBuild
✓ Succeeded - 7 minutes ago
Details

DestroyStack
AWS Lambda
✓ Succeeded - 6 minutes ago
Details

DeleteTestStack
AWS CloudFormation
✓ Succeeded - 6 minutes ago
Details

Source: fix: cypress project ccb434a5

Key features

- Infrastructure as Code
- Each Git branch has its own **independent stack**
- **No shared resources** between different feature stacks
- CloudFormation **automatically** updates a Stack by inserting or deleting resources
    - **No need** for manual resource management
- Pay only when the service is used
- **Automatic scaling** depending on the demand

Infrastructure as Code

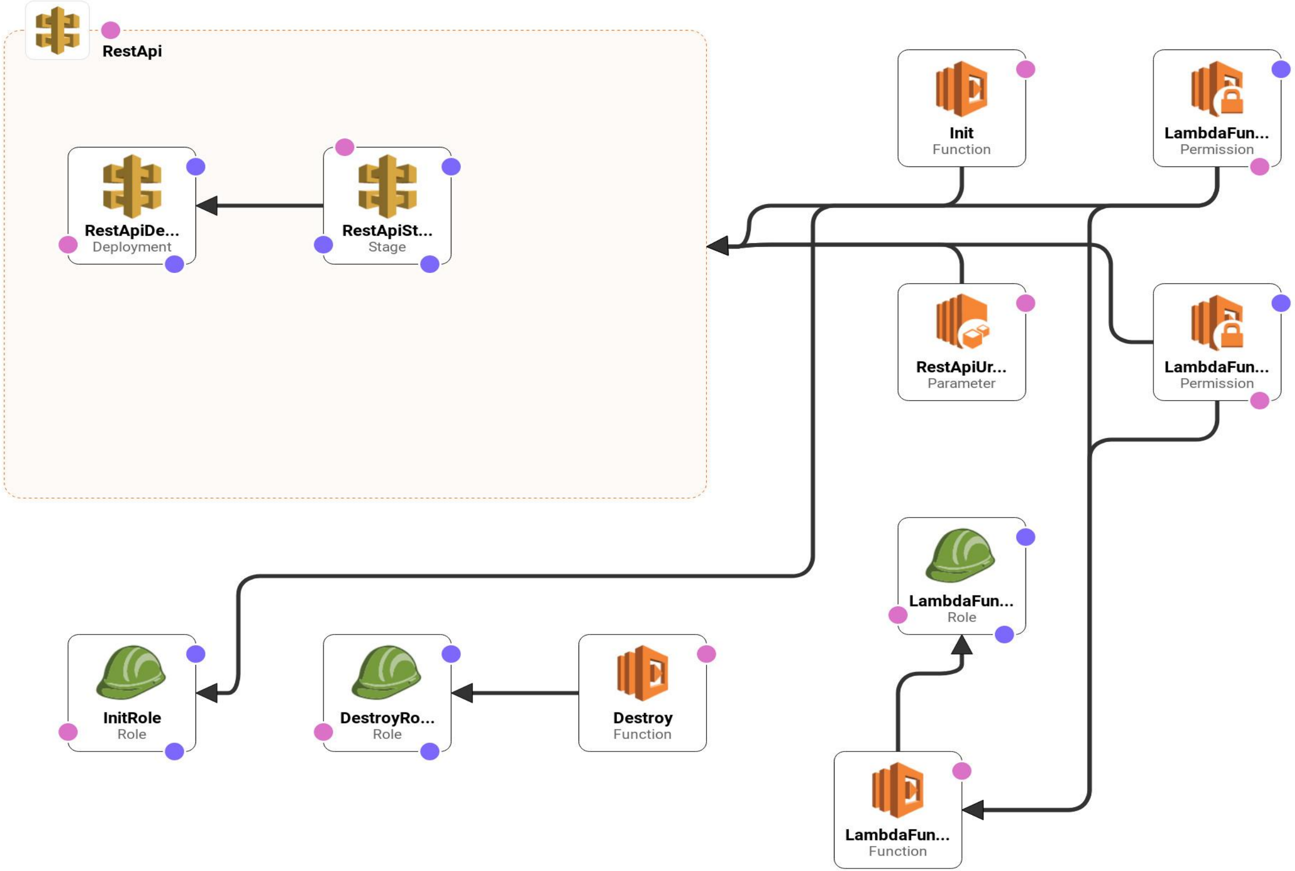
```yaml
Resources:
  RestApi:
    Type: AWS::Serverless::Api
    Properties:
      StageName: !Ref Stage
      DefinitionBody:
        Fn::Transform:
          Name: 'AWS::Include'
          Parameters:
            Location: !Join ['', ['s3://', !Ref 'CodeBucket', '/openapi.yaml']]
  LambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: no.bibsys.handlers.StreamLambdaHandler::handleRequest
      Runtime: java8
      CodeUri: api/build/libs/api-fat.ja
      Events:
        ApiResource:
          Type: Api
          Properties:
            Path: /{proxy+}
            Method: any
            RestApiId: !Ref RestApi
```

MEET SAM.

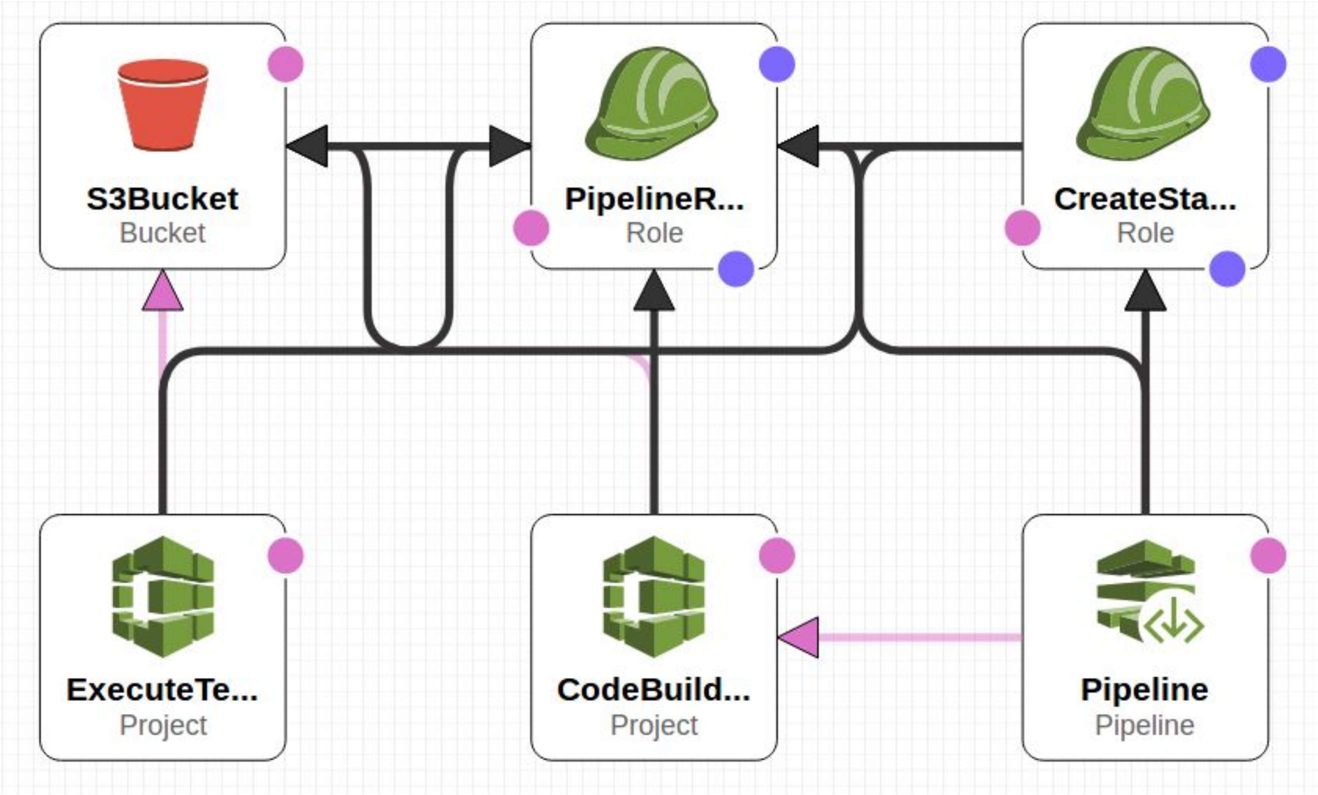
USE SAM TO BUILD TEMPLATES THAT DEFINE YOUR SERVERLESS APPLICATIONS.

DEPLOY YOUR SAM TEMPLATE WITH AWS CLOUDFORMATION.

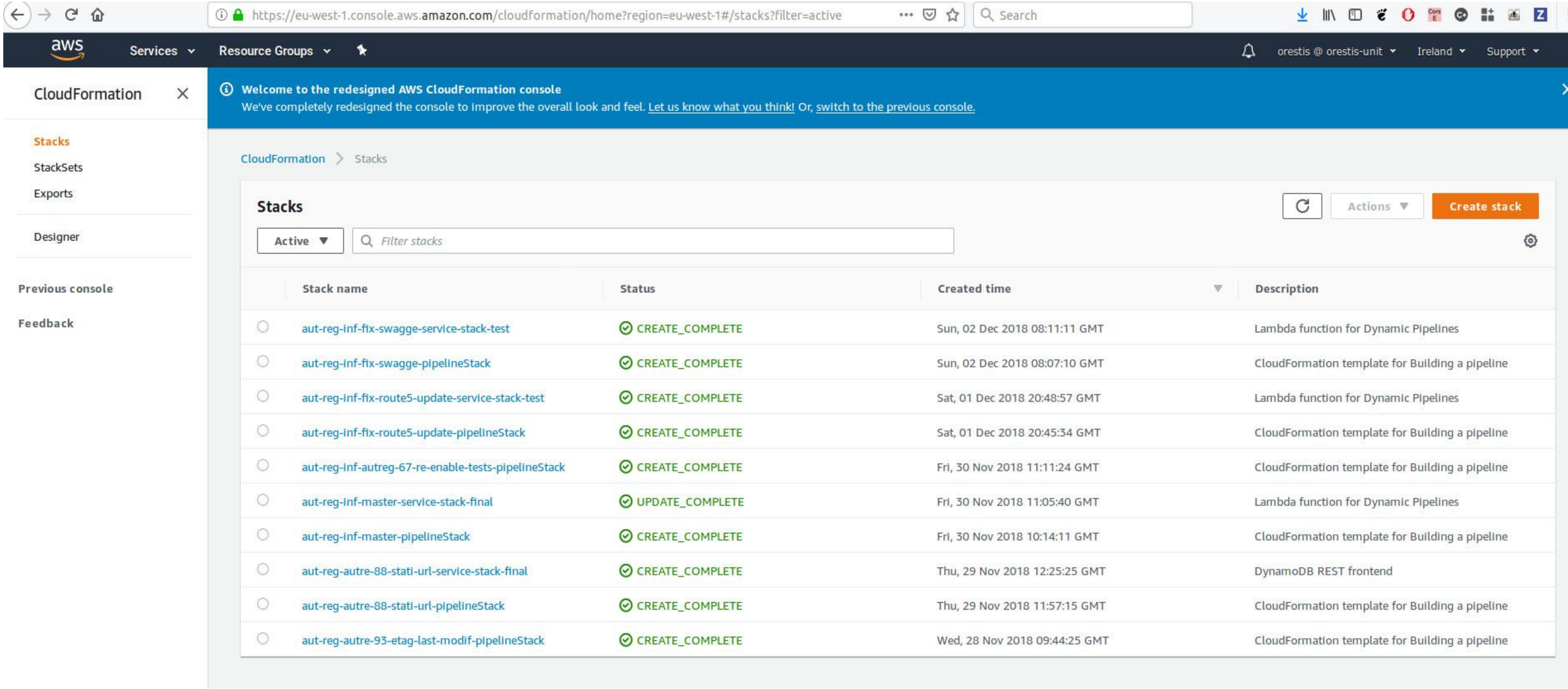
CloudFormation Application Stack



CloudFormation Pipeline Stack



CloudFormation Stacks



Welcome to the redesigned AWS CloudFormation console
We've completely redesigned the console to improve the overall look and feel. Let us know what you think! Or, switch to the previous console.

Stacks

	Stack name	Status	Created time	Description
○	aut-reg-inf-fix-swagge-service-stack-test	✓ CREATE_COMPLETE	Sun, 02 Dec 2018 08:11:11 GMT	Lambda function for Dynamic Pipelines
○	aut-reg-inf-fix-swagge-pipelineStack	✓ CREATE_COMPLETE	Sun, 02 Dec 2018 08:07:10 GMT	CloudFormation template for Building a pipeline
○	aut-reg-inf-fix-route5-update-service-stack-test	✓ CREATE_COMPLETE	Sat, 01 Dec 2018 20:48:57 GMT	Lambda function for Dynamic Pipelines
○	aut-reg-inf-fix-route5-update-pipelineStack	✓ CREATE_COMPLETE	Sat, 01 Dec 2018 20:45:34 GMT	CloudFormation template for Building a pipeline
○	aut-reg-inf-autreg-67-re-enable-tests-pipelineStack	✓ CREATE_COMPLETE	Fri, 30 Nov 2018 11:11:24 GMT	CloudFormation template for Building a pipeline
○	aut-reg-inf-master-service-stack-final	✓ UPDATE_COMPLETE	Fri, 30 Nov 2018 11:05:40 GMT	Lambda function for Dynamic Pipelines
○	aut-reg-inf-master-pipelineStack	✓ CREATE_COMPLETE	Fri, 30 Nov 2018 10:14:11 GMT	CloudFormation template for Building a pipeline
○	aut-reg-autre-88-stati-url-service-stack-final	✓ CREATE_COMPLETE	Thu, 29 Nov 2018 12:25:25 GMT	DynamoDB REST frontend
○	aut-reg-autre-88-stati-url-pipelineStack	✓ CREATE_COMPLETE	Thu, 29 Nov 2018 11:57:15 GMT	CloudFormation template for Building a pipeline
○	aut-reg-autre-93-etag-last-modif-pipelineStack	✓ CREATE_COMPLETE	Wed, 28 Nov 2018 09:44:25 GMT	CloudFormation template for Building a pipeline

Summary

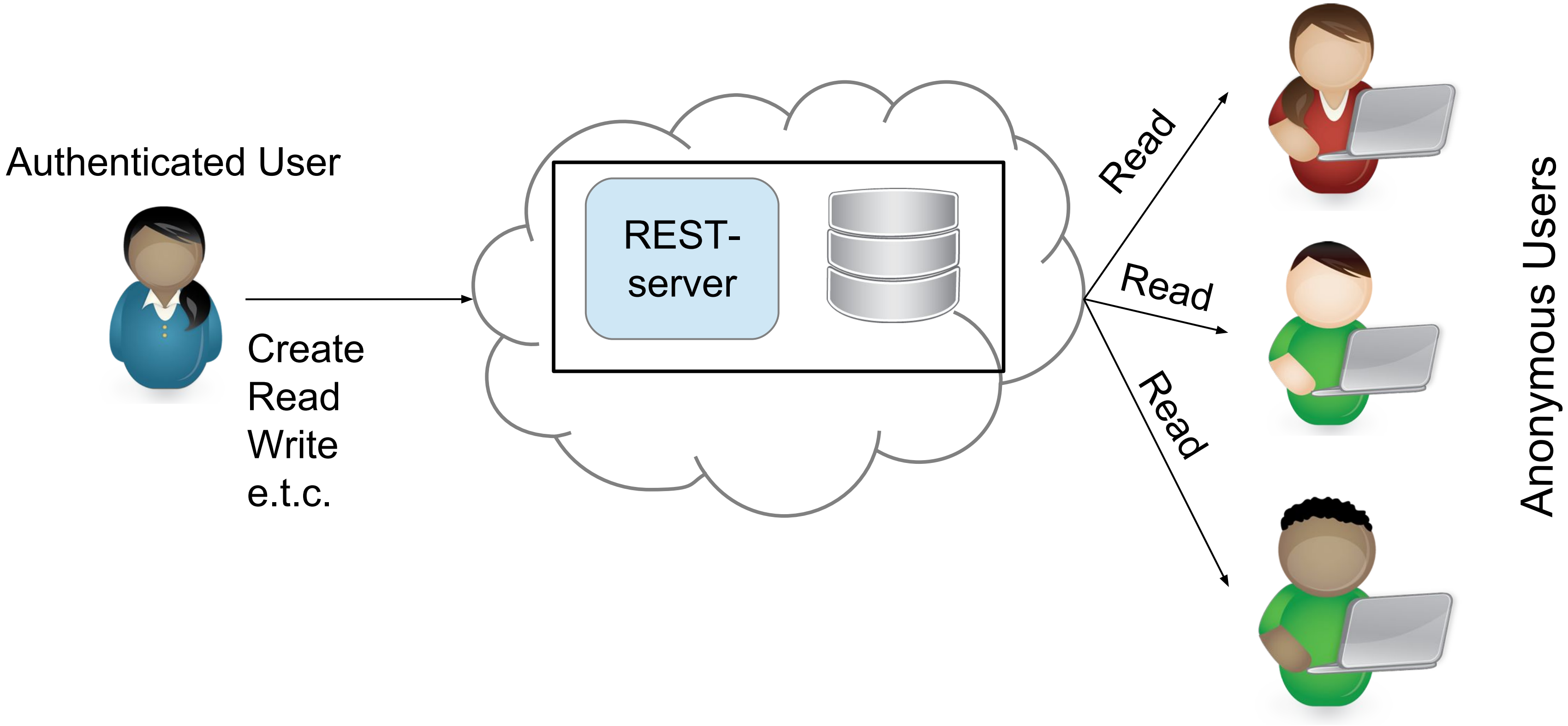
- Developed an Entity-registry where each client can have **multiple registries** with **dynamically** specified data structured
- Used Gherkin to formalize the communication between the customer, the product owner, and the developers
- **Formal definition** when the product will be **ready** (Gherkin)
- **Minimized** the maintenance needs using **AWS Serverless technologies**
- **Created a general purpose library** for deploying services/applications in the AWS platform (publicly available :)

Thank you!

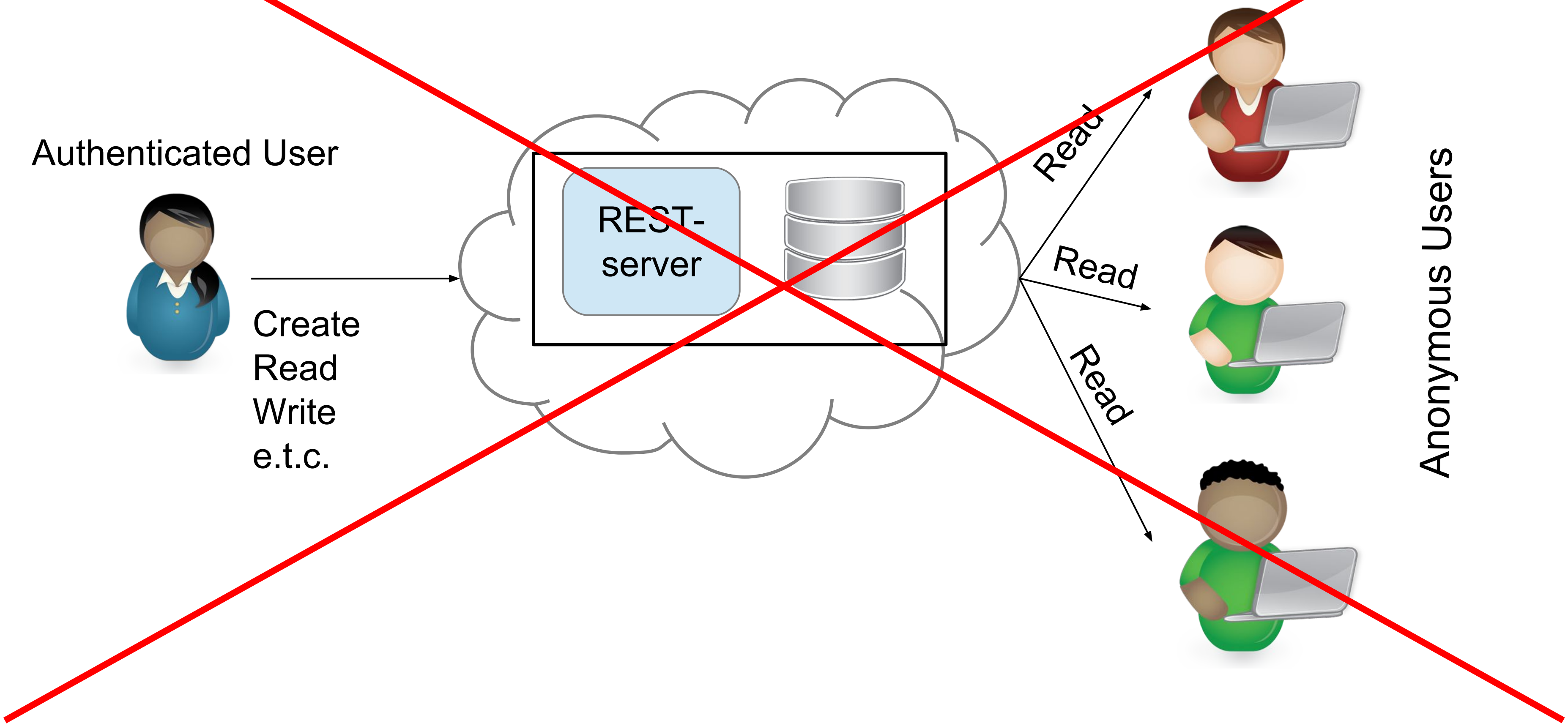
Questions?

Buffer

OK! I think I know what to do...



OK! I think I know what to do...


Authenticated User
Create
Read
Write
e.t.c.
REST-server
Read
Read
Read
Anonymous Users