Infrastructure as Code: "pip install" your environment

Sebastian Neubauer
@sebineubauer

blue**yonder**

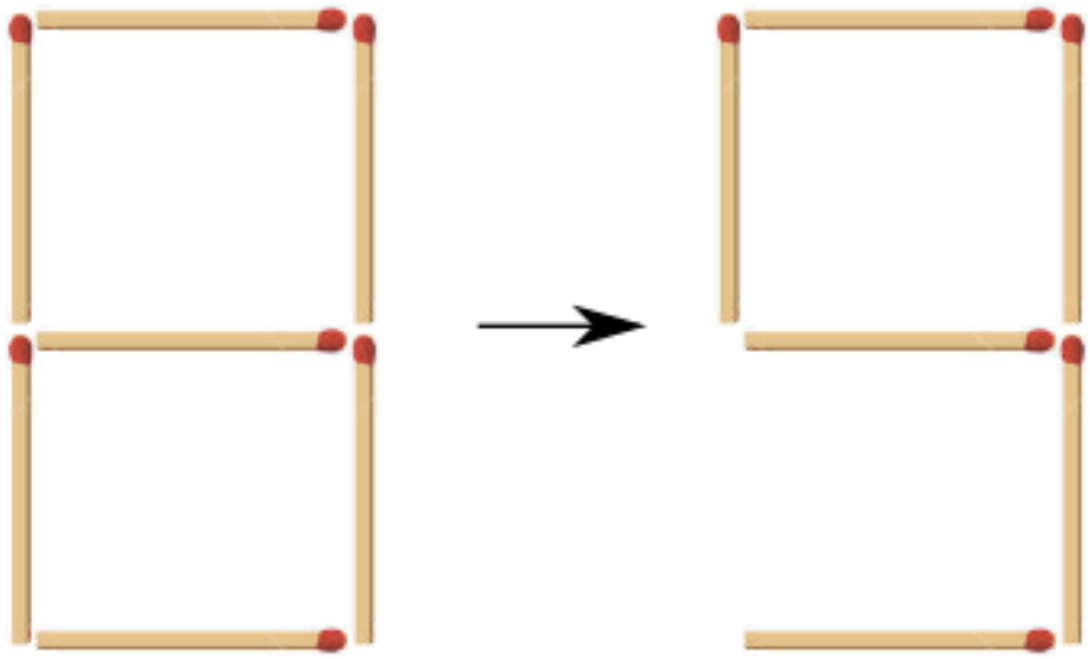
Outline

- What is **CRUD** and what has it to do with immutability?
- What is infrastructure as **code**?
- A real world example: **Postgraas** - PostgreSQL-as-a-service
- What **you** can do and where to start?
- Summary

What is CRUD?

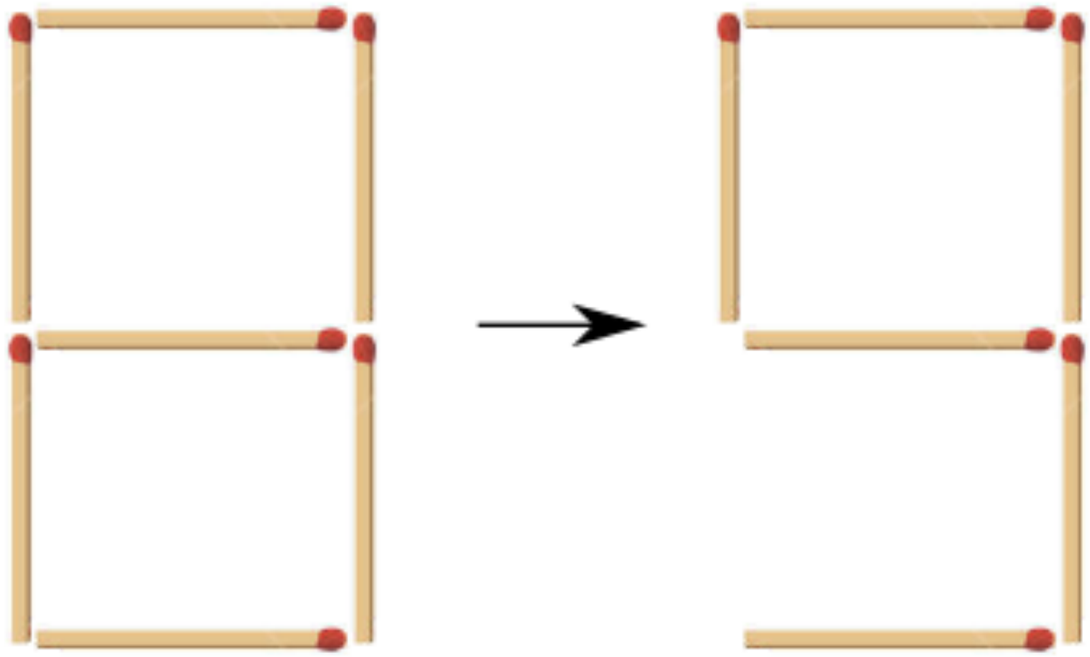
- create, read, update, delete
- this simple set of operations is really powerful
- nearly everything can be implemented using these operations:
  - Blogs, todo app, ec2 instances, dictionaries, ldap users…
- If implemented with these, you get a REST api basically for free:
  - POST, GET, PUT, DELETE

What is immutability?



What do we have to do to come from left to right?

What is immutability?



How would a machine do it?

What is immutability?

- An update can always be implemented by deleting and recreating
- deleting and recreating in the real world is often resource intensive:
  - building a house, writing a book, provision a server…
- making an update can be quite difficult and complex
  - add a room to a house, change the structure of a database, change the hair color of Mona Lisa…

What is immutability?

- Humans are lazy but intelligent

  👉Update is most often the way to go

- Machines are eager but dumb

  👉deleting and recreating is the preferred way to go

What is the current status of software infrastructure?

What is immutability?

„Immutability is a concept of how to change the state of a system“

@sebineubauer

What is immutability?

API crash course:
From CRUD to REST

- collection resource:
  - uri: /instances/
  - GET: list all instances
  - POST: create new instance
- instance resource:
  - uri: /instances/ID
  - GET: details of the instance
  - DELETE: delete instance
  - PUT: change the instance

what is infrastructure?

„Infrastructure is everything what brings your dead code to life“

what is infrastructure?

Deploy

Object stores

Logging

Caches

„Infrastructure is everything what brings your dead code to life“

Compute resources

Key-value stores

Middleware

Routing

Databases

Identity access

Network

Application Server

What is infrastructure as code?

- …if we can access infrastructure in an automated fashion via machine consumable APIs:
  - no tickets
  - no screwdrivers
  - no admin ssh-terminal
  - no GUI
  - …

What is infrastructure-as-a-service?

- …if the application itself can consume the infrastructure it runs in:
  - self service
  - simple, machine consumable api
  - identity access management
  - no dependency to screwdrivers, admins or stackoverflow deploy instructions, …

What is immutable infrastructure?

What is immutable infrastructure?

Why is automation so important?

- automation means: machines do it
  - reduced human error
  - reproducibility
  - cheaper
  - faster
  - happier developers

Why is X-as-a-service so important?

„Now we can put the entire application, including infrastructure under version control“

@sebineubauer

Now let’s get our hands dirty!

Let’s build a PostgreSQL-as-a-service service together!

What should it do

- provide PostgreSQL instances-as-a-service
- isolate these instances as good as possible
- provide self service
- provide a simple, machine consumable API
- persist the status of the instances in a „meta db“
- be as simple as possible

First we need a name:

Brainstorm a possible implementation

How can we provide instant PostgreSQL?

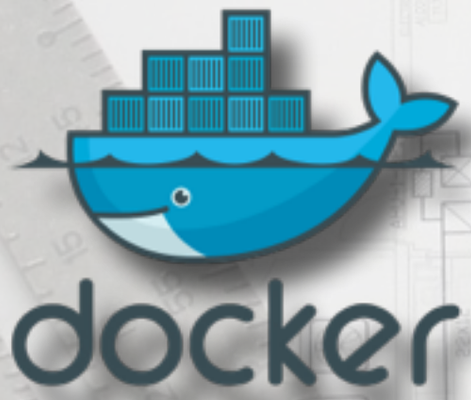
How should the api look like?

How do we implement the api?

@sebineubauer

Brainstorm a possible implementation

How can we provide instant PostgreSQL?

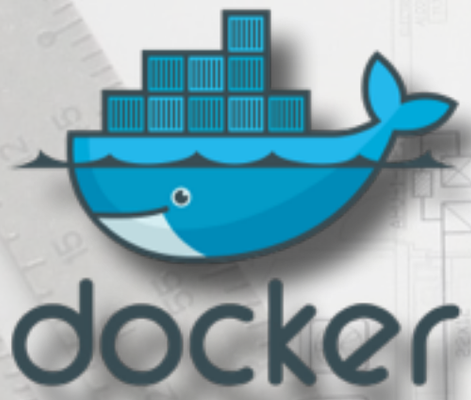


How should the api look like?

How do we implement the api?

Brainstorm a possible implementation

How can we provide instant PostgreSQL?



How should the api look like?

RESTful API

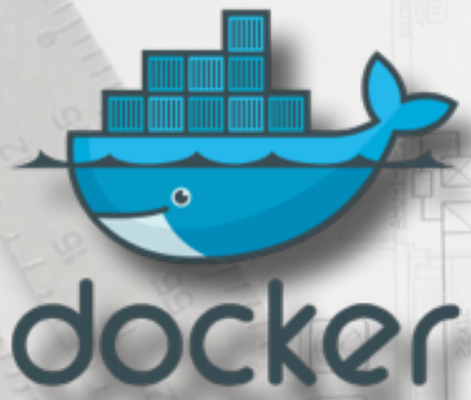
GET PUT POST DELETE

How do we implement the api?

@sebineubauer

Brainstorm a possible implementation

How can we provide instant PostgreSQL?



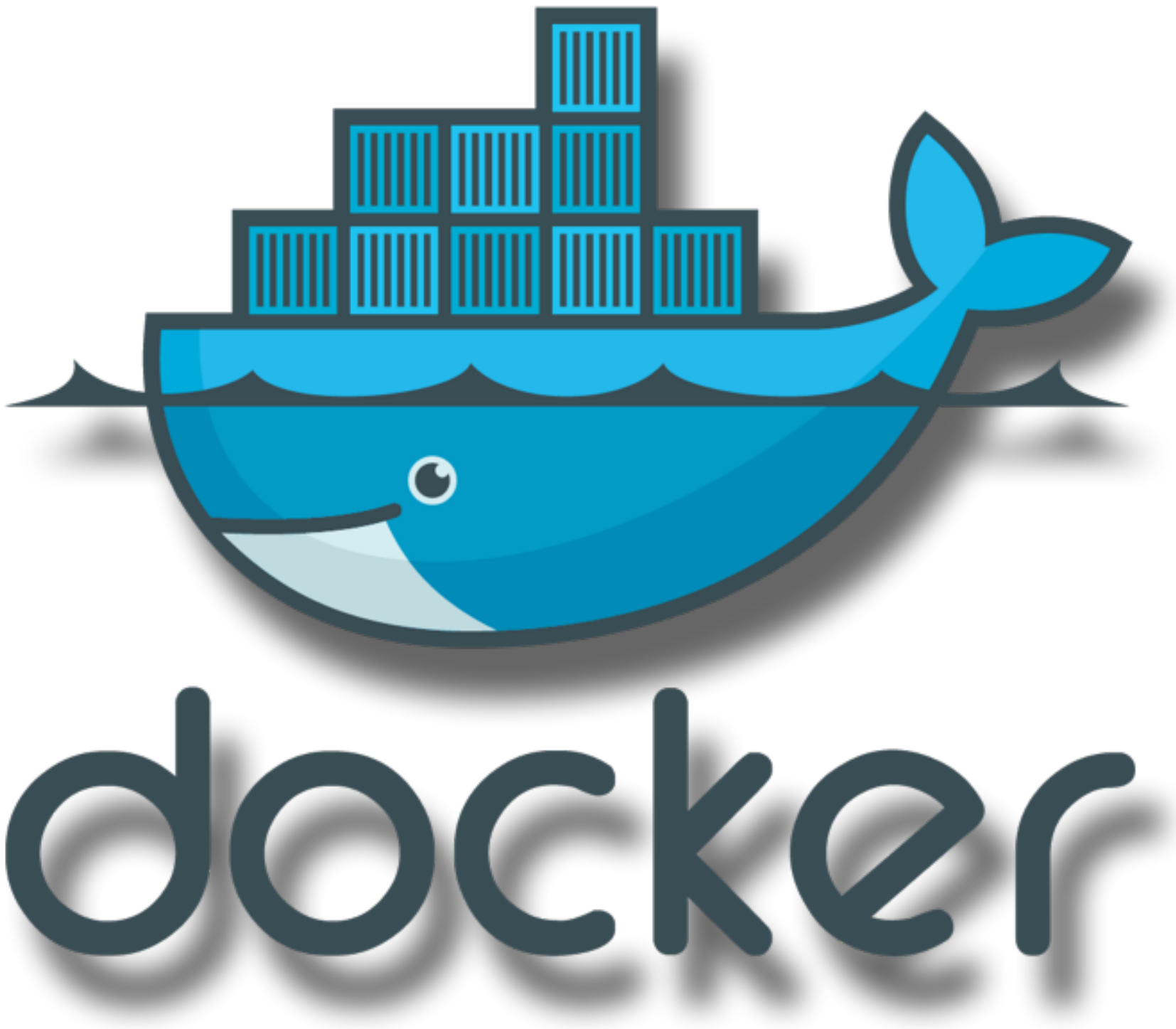
How should the api look like?

RESTful API

GET PUT POST DELETE

How do we implement the api?

@sebineubauer



@sebineubauer

Managing docker from Python

- Easy with docker client library docker-py
- Nearly all features accessible from python
- Needs a running docker daemon to connect to:
  - remote or local
- Quite good quality (e.g. compared to the kafka python client)

Creating a PostgreSQL instance

```
def create_postgres_instance(postgraas_instance_name, connection_dict):
    c = docker.Client(base_url='unix://var/run/docker.sock')
    environment = {
        "POSTGRES_USER": connection_dict['db_username'],
        "POSTGRES_PASSWORD": connection_dict['db_pwd'],
        "POSTGRES_DB": connection_dict['db_name']
    }
    internal_port = 5432
    if check_container_exists(postgraas_instance_name):
        raise ValueError('Container exists already')
    image = 'postgres'
    container_info = c.create_container(image,
                                        name=postgraas_instance_name,
                                        ports=[internal_port],
                                        environment=environment,
                                        labels={"postgraas": image})
    container_id = container_info['Id']
    port_dict = {internal_port: connection_dict['port']}
    c.start(container_id, port_bindings=port_dict)
    return container_id
```

Get details of an instance

```
def get_container_by_name(postgraas_instance_name):
    c = docker.Client(base_url='unix://var/run/docker.sock')
    containers = c.containers()
    for container in containers:
        for name in container['Names']:
            if postgraas_instance_name in name.replace("/", ""):
                return container
```

Delete a PostgreSQL instance

(Yes, that is all we need…)

```
def delete_postgres_instance(container_id):
    c = docker.Client(base_url='unix://var/run/docker.sock')
    c.remove_container(container_id, force=True)
    return
```

RESTful API

GET PUT POST DELETE

@sebineubauer

Design the API

…easy, we know already…

- Create an instance:
  - POST /postgraas_instances
  - parameters:
    - postgraas_name
    - db_name
    - db_username
    - db_pwd
- returns: instanceID

Design the API

- List all instances:
  - GET /postgraas_instances
  - returns: list of all instances (or instanceID)
- Get an instance:
  - GET /postgraas_instances/<instanceID>
  - returns: instance details
- Delete an instance:
  - DELETE /postgraas_instances/<instanceID>
  - returns: success

Flask

web development,
one drop at a time

@sebineubauer

Implement REST in Python

- Really easy using the Flask extension flask-restful
- Define a resource by deriving from a `Resource` class
- Implement members `get`, `put`, `post` or `delete`
- set URI's for each defined resource
- Use SQLAlchemy for the meta data persistance

The Collection Resource

```
db_instance_marshaller = {
    'id': fields.Integer,
    'postgraas_instance_name': fields.String,
    'creation_timestamp': fields.DateTime(dt_format='iso8601'),
    'db_name': fields.String,
    'username': fields.String,
    'password': fields.String,
    'hostname': fields.String,
    'port': fields.String,
    'container_id': fields.String,
}

class DBInstanceCollectionResource(Resource):

    @marshal_with(db_instance_marshaller)
    def get(self):
        all = DBInstance.query.all()
        return all
```

```
def post(self):
    parser = reqparse.RequestParser()
    parser.add_argument('postgraas_name', required=True, type=str, help='name')
    parser.add_argument('db_name', required=True, type=str, help='name of the db')
    parser.add_argument('db_username', required=True, type=str, help='username of the db')
    parser.add_argument('db_pwd', required=True, type=str, help='pass of the db user')
    args = parser.parse_args()
    db_credentials = {
        "db_name": args['db_name'],
        "db_username": args['db_username'],
        "db_pwd": args['db_pwd'],
        "host": pg.get_hostname(),
        "port": pg.get_open_port()
    }
    if DBInstance.query.filter_by(postgraas_instance_name=args['postgraas_name']).first():
        return {'msg': "postgraas_name already exists {}".format(args['postgraas_name']) }
    try:
        db_credentials['container_id'] = pg.create_postgres_instance(args['postgraas_name'],
                                                                     db_credentials)
    except APIError as e:
        return {'msg': str(e)}
    db_entry = DBInstance(postgraas_instance_name=args['postgraas_name'],
                          db_name=args['db_name'],
                          username=args['db_username'],
                          password="",
                          hostname=db_credentials['host'],
                          port=db_credentials['port'],
                          container_id=db_credentials['container_id'])
    db.session.add(db_entry)
    db.session.commit()
    db_credentials["postgraas_instance_id"] = db_entry.id
    return db_credentials
```

The Instance Resource

```
class DBInstanceResource(Resource):

    @marshal_with(db_instance_marshaller)
    def get(self, id):
        entity = DBInstance.query.get(id)
        return entity
```

```
def delete(self, id):
    c = docker.Client(base_url='unix://var/run/docker.sock',
                      version='auto',
                      timeout=10)
    entity = DBInstance.query.get(id)
    if entity:
        try:
            container_info = c.inspect_container(entity.container_id)
            print container_info
        except APIError as e:
            if e.response.status_code == 404:
                db.session.delete(entity)
                db.session.commit()
                return {'status': 'success', 'msg': 'deleted'}
        try:
            pg.delete_postgres_instance(entity.container_id)
        except APIError as e:
            return {'status': 'failed', 'msg': str(e)}
        db.session.delete(entity)
        db.session.commit()
        return {'status': 'success', 'msg': 'deleted postgraas instance'}
    else:
        return {'status': 'failed', 'msg': 'does not exist {}'.format(id)}
```

Finally: set URIs for the Resources

```
from flask import Flask
from flask_restful import fields, Resource, marshal_with, Api, reqparse
from postgraas_server.management_resources import db
from postgraas_server.management_resources import DBInstanceResource,
DBInstanceCollectionResource
from postgraas_server.configuration import get_meta_db_config_path


def create_app(config):
    app = Flask(__name__)
    app.config['SQLALCHEMY_DATABASE_URI'] = get_meta_db_config_path(config)
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    restful_api = Api(app)
    restful_api.add_resource(DBInstanceResource, "/postgraas_instances/<int:id>")
    restful_api.add_resource(DBInstanceCollectionResource, "/postgraas_instances")
    db.init_app(app)
    return app
```

Done! We have all parts together…

And does it work?

1. Starting Docker and pull the `postgres` image
2. Starting up the flask server (e.g. gunicorn)
3. Initialize the meta db
4. and then…

KEEP CALM AND GET YOU A POSTGRES

@sebineubauer

my_postgraas.json:

```
{
   "postgraas_name": "my_postgraas",
   "db_name": "my_db",
   "db_username": "db_user",
   "db_pwd": "secret"
}
```

```
~$ curl -H "Content-Type: application/json" \
    -X POST \
    –data @my_postgraas.json \
    http://localhost:8080/postgraas_instances

{
    "postgraas_instance_id": 1,
    "container_id": 193f0d94d49fa26626fdbdb583e9207b4852831a105c03",
    "db_pwd": "secret",
    "host": "not implemented yet",
    "db_name": "my_db",
    "db_username": "db_user",
    "port": 54648
}
~$ psql -h localhost -p 54648 -U db_user my_db

Welcome to psql 9.5.3, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \q to quit

my_db =>
```

Postgraas Summary

- It is on github, contributions very welcome: https://github.com/blue-yonder/postgraas_server
- Running in our company since a year:
  - for integrations tests
  - for experiments
  - maybe soon for productive services
  - served over hundreds of postgres instances, 50 currently running

Now it is your turn…

- Take a manual step or a ticket in your delivery chain
  Example: A directory on a fileserver with some permissions
- Blueprint:
  - Flask with REST API
  - 'os.makedirs' and 'shutil.rmtree' for create and delete
- Design API:
  - Needed: name, linux user, permissions, quota?
- Implement:
  - Needed: A rainy Sunday afternoon!

Summary/TIL

- CRD is for machines, CRUD is for humans
- CRD is trivial to be implemented as REST API
- Most infrastructure can be expressed as CRD:
  - can be easily exposed via REST API
  - is immutable, so good for machines
  - can be consumed from applications as a self service
- Postgraas is a nice and useful example, that the above is true…

Thank You!