JRA 2/3 Infrastructure Management Workshop

Seemless RI integration using declarative configuration management

Steffen Vogel

Institute for Automation of Complex Power Systems

RWTH Aachen University, Aachen, Germany

26th January 2022

EU H2020 Programme GA No. 870620

● The workshop will be recorded

Agenda

1. Problem Statement
   - Objectives for today
2. Infrastructure as Code
   - Declarative Configuration Management
3. Kubernetes
   - Containers
   - Interfaces
   - Operators & Reconciliation
4. RIasC
   - Provisioning
   - Virtual Networking
   - Network Emulation
   - Network Monitoring
   - Time-synchronization
   - Multi Tenancy
5. Operator Example
6. Feedback, Discussion & Questions

Duration:
1-2 hrs

Problem & Challenges

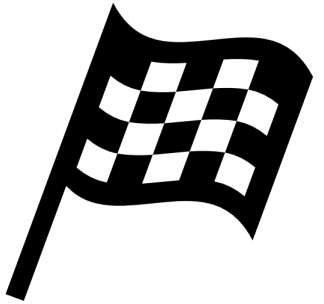
- Infrastructure Management
  - Complexity
  - Reproducability
  - Drift
  - Resource Sharing
  - Remote Access & Remote Preperation

Objectives of this Workshop

- Present JRA2.2.5 development "*Accelerate time-to-experiment for Remote RI*"

- Introduce the basics of Kubernetes & declartive configuration management
  - The operator reconciliation loop

- Provide inputs and ideas for ERIGrid's JRA3 platform
  - RIasC - Research Infrastructure as Code



Inspiration: Infrastructure as Code

“**Infrastructure as code** (**IaC**) is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

The infrastructure managed by this process comprises both physical equipment, as well as software tools. The definitions may be in a version control system. It can use either scripts or declarative definitions, rather than manual processes, but the term is more often used to promote declarative approaches.”

Source: Wikipedia

EU H2020 Programme GA No. 870620

Research Infrastructure as Code (RIasC)

“***Research*** **Infrastructure as code** (**RIasC**) is the process of managing and provisioning *research experiments* through machine-readable definition files, rather than manual hardware configuration or interactive configuration tools.

Experiments managed by this process comprises both physical equipment, as well as software tools. The definitions may be in a version control system. It can use either scripts or declarative definitions, rather than manual processes, but the term is more often used to promote declarative approaches.”

EU H2020 Programme GA No. 870620

Existing tools for IaC / Configuration Management

- Puppet (2005)
- Chef (2009)
- SaltStack (2011)
- Ansible (2012)
- Terraform (2014)
- Kubernetes (2015)

Container Orchestration & more?

KUBERNETES

Kubernetes

- Container Orchestration System
  - Executes of containerized applications on a cluster of one or multiple machines
  - Uses declarative configuration for cluster state
- Open Sourced by Google in 2015
  - Now managed by Cloud Native Computing / Linux Foundation
  - ~4 releases per year
  - Written in Go programming language
- Highly extensible
  - Plugins / Operators for Storage, Networking, and much more
  - Custom Ressources

Containers at Google

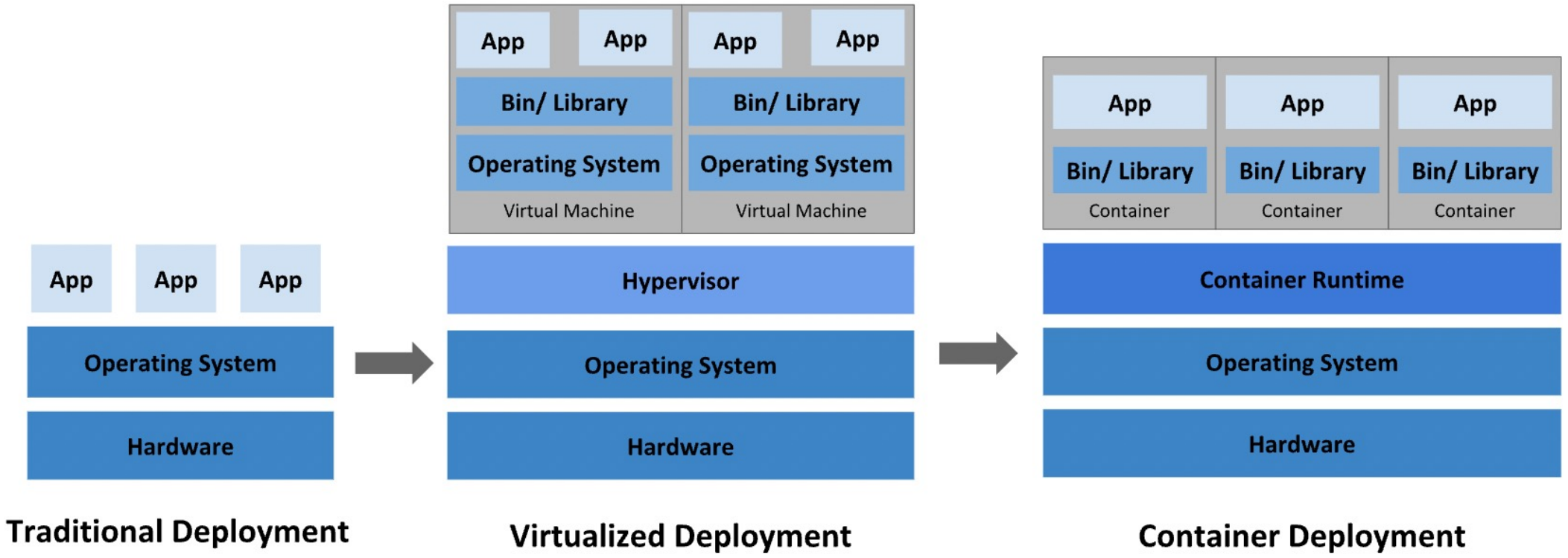
"Everything at Google runs in a Container"

"We start over 2 billion containers per week"

(around 3000 per second)

Source: "Joa Beda: Containers as Scale, GlueCon, May 22 2014"

Evolution of software deployment



Source: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Containers & Images

- Packages software applications in an **exchangable** and **reproducible** format
- Large library of existing images for existing software
- Dockerfile's as receipes for building a Container image:

Dockerfile

```
FROM python:3.8

WORKDIR /
ADD requirements.txt main.py /
RUN pip3 install -r requirements.txt

CMD ["python3", "/main.py"]
```

Test locally:

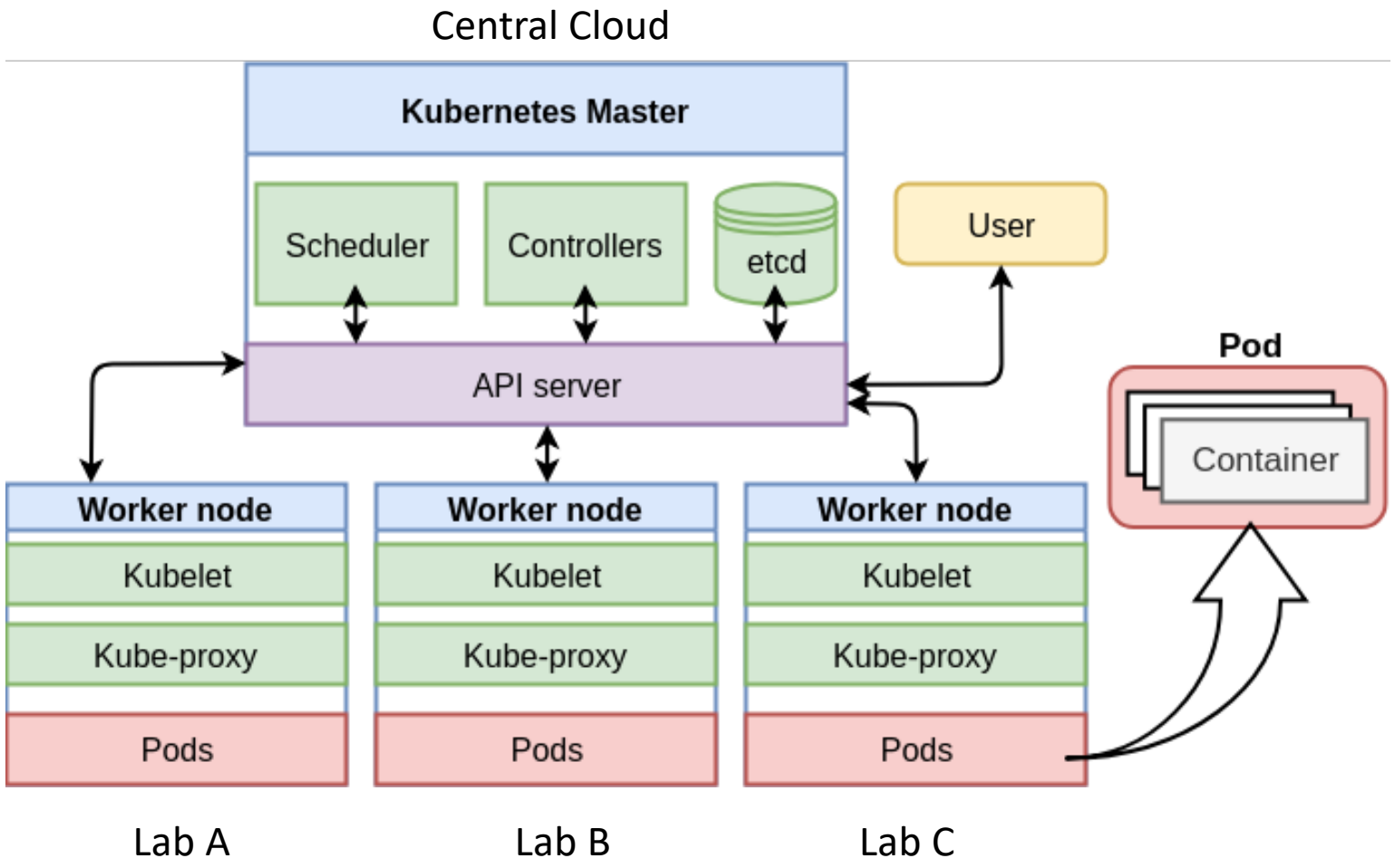
```
docker build -t erigrid/py_ctrl:v1 -f Dockerfile .
docker run erigrid/py_ctrl:v1
```

Publish or get to image registry:

```
docker push erigrid/my_py_ctrl:v1
docker pull erigrid/my_py_ctrl:v1
```

ERIGrid Images: https://hub.docker.com/orgs/erigrid/repositories

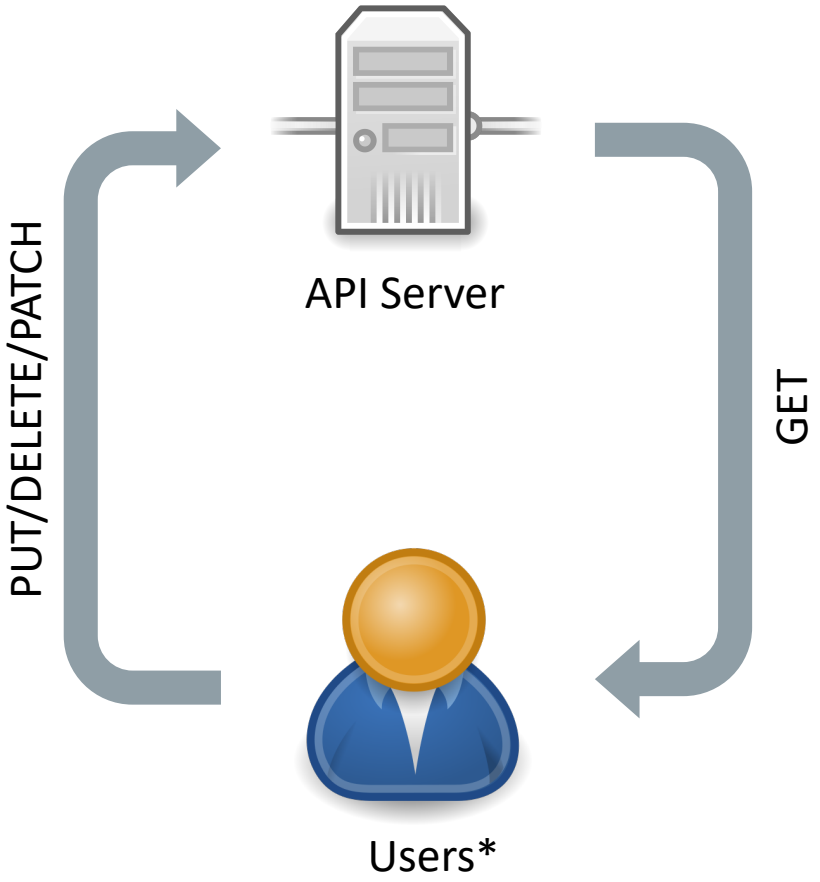
Architecture



Source: “Olle Larsson, Running Databases in a Kubernetes Cluster, M.Sc. Thesis, 2019, Umea University”

Manifests

- Each resource in Kubernetes is described by a manifest
  - Represents the **desired** state
  - Many types of resources are available:
    - Workloads: Containers, Pods
    - Storage: Storage Volumes, Configuration Files, Secrets
    - Network: Load Balancers, Autoscaling …
- They are stored by Kubernetes API server
- Usually formatted as JSON or YAML documents



Manifests: Basic Structure

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: py_ctrl
  labels:
    app: controller
    project: eg2-ta-123
  annotations:
    last-change: “2022-01-24 11:01:01"
spec:
  [...]
```

YAML Document Separator
(when describing multiple resources in a single file)

Preamble
(same for all resources)

Resource Specification
(structure depends on **kind** field)

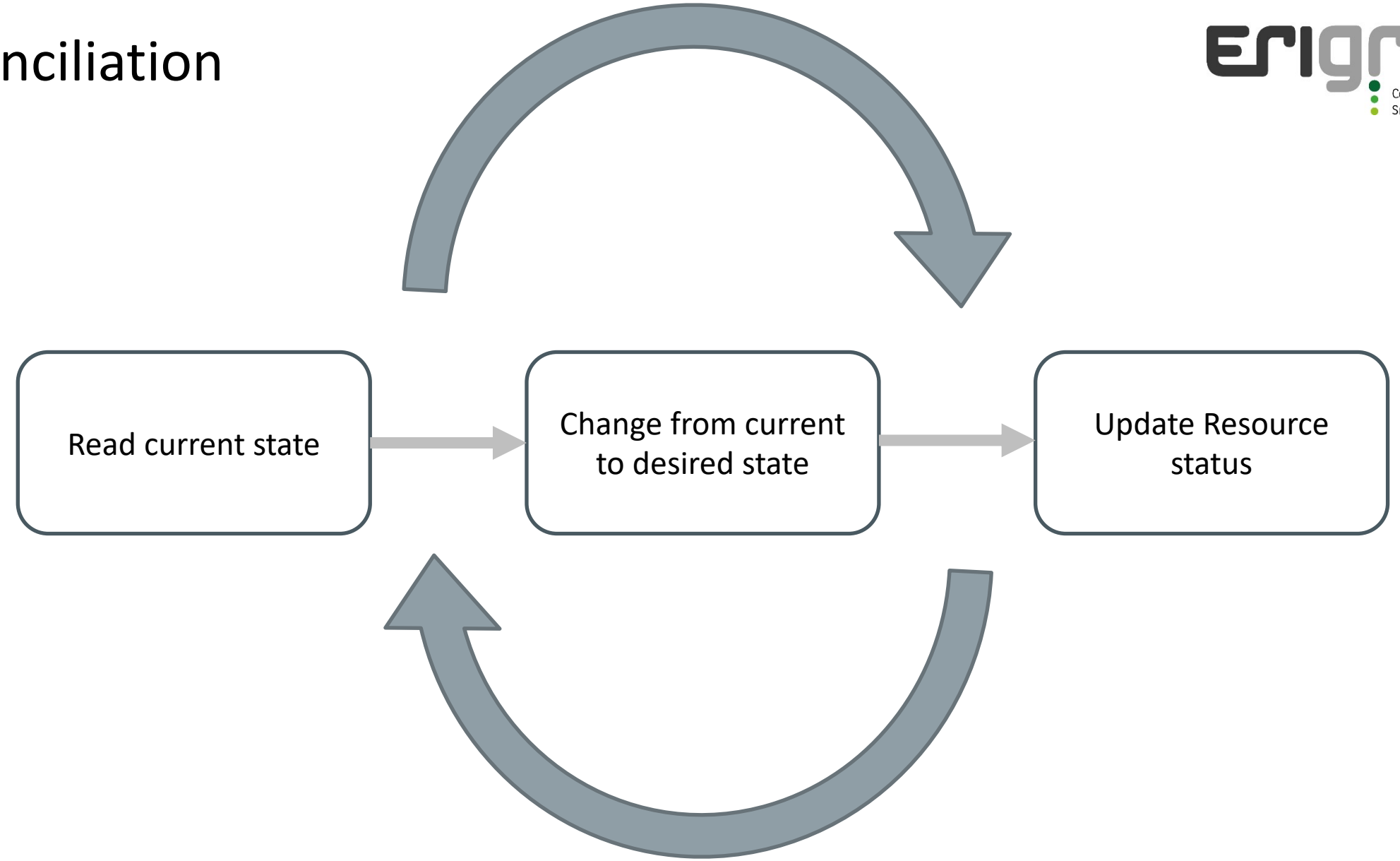
EU H2020 Programme GA No. 870620

Manifests: Examples

```yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: py_ctrl
  labels:
    app: controller
    project: eg2-ta-123
  annotations:
    last-change: “2022-01-24 11:01:01"
spec:
  nodeSelector:
    institution: rwth
  containers:
  - name: controller
    image: erigrid/py_ctrl
```

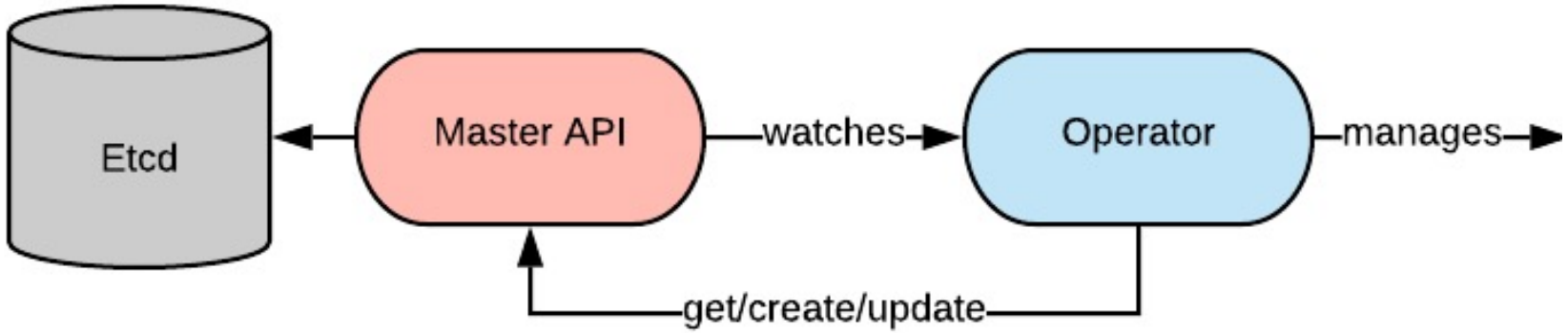
```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: results
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 8Gi
```

Reconciliation



Custom Resource Descriptions (CRD)

- We can define our own resources types
  - And logic (operators) which processes them

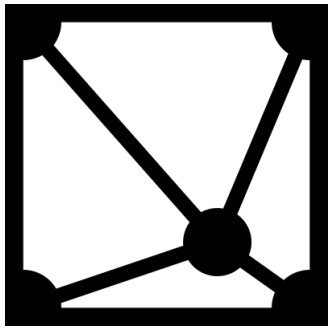


Source: https://cloud.redhat.com/blog/kubernetes-operators-best-practices

Interfaces

- **Web**
  - Kubernetes Dashboard
  - Rancher UI
- **CLI:** Command Line
  - `kubectl`
  - `helm`
  - `curl`
- **API:** REST / Language Bindings
- **IDE:** Development Environments
  - Lens
  - Visual Studio Code
- **Custom**?

Kubernetes REST API Specification:
https://petstore.swagger.io/?url=https://raw.githubusercontent.com/kubernetes/kubernetes/master/api/openapi-spec/swagger.json#/



Connecting European Smart Grid Infrastructures

Research Infrastructure as Code

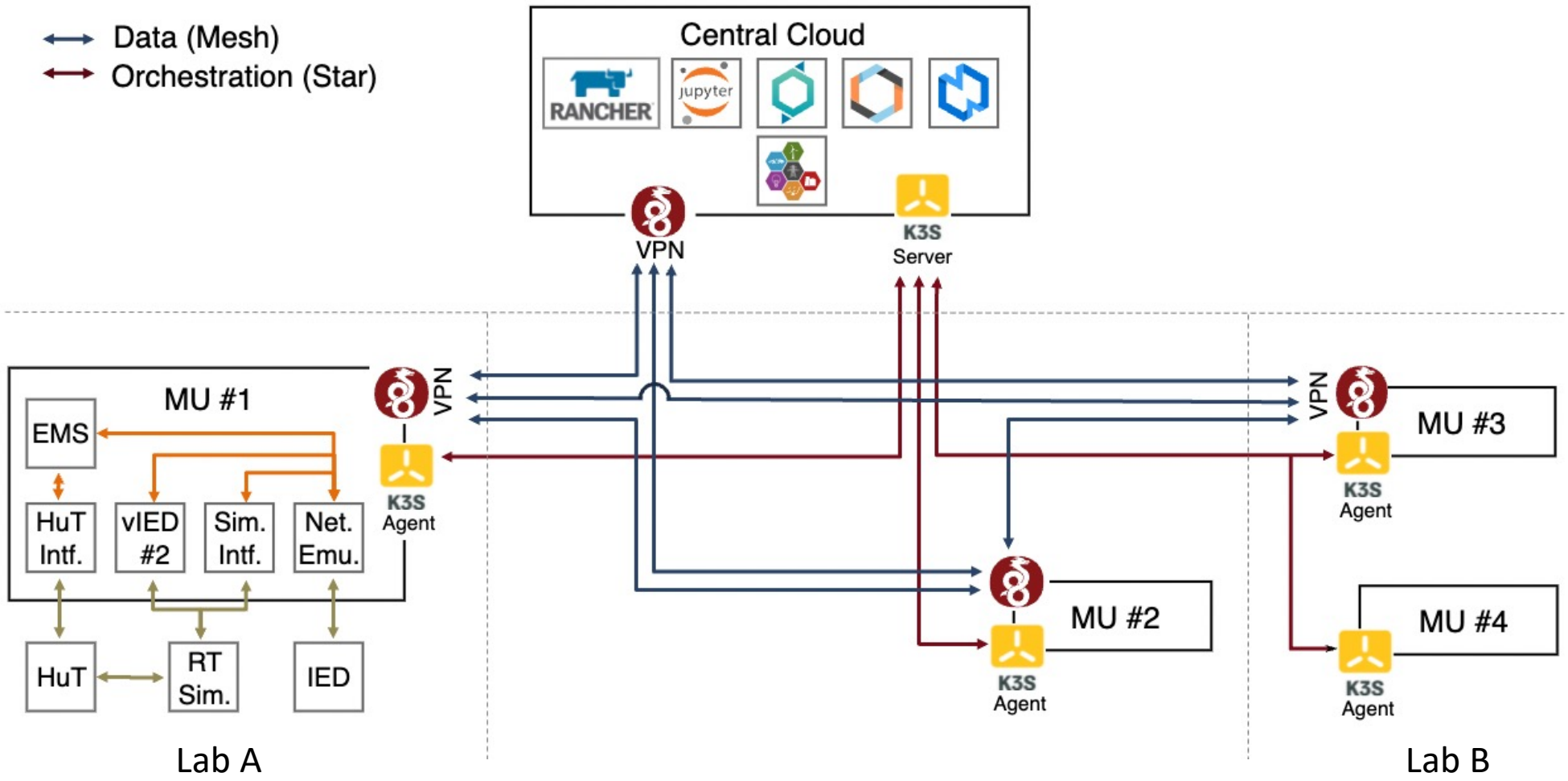
RIasC

riasc.eu

RIasC

- RIasC is a set of tools to run research experiments on a (distributed) Kubernetes cluster
  - Opinionized selection of Kubernetes addons
    - Networking, Storage
  - Custom Resources
  - Operators

Architecture

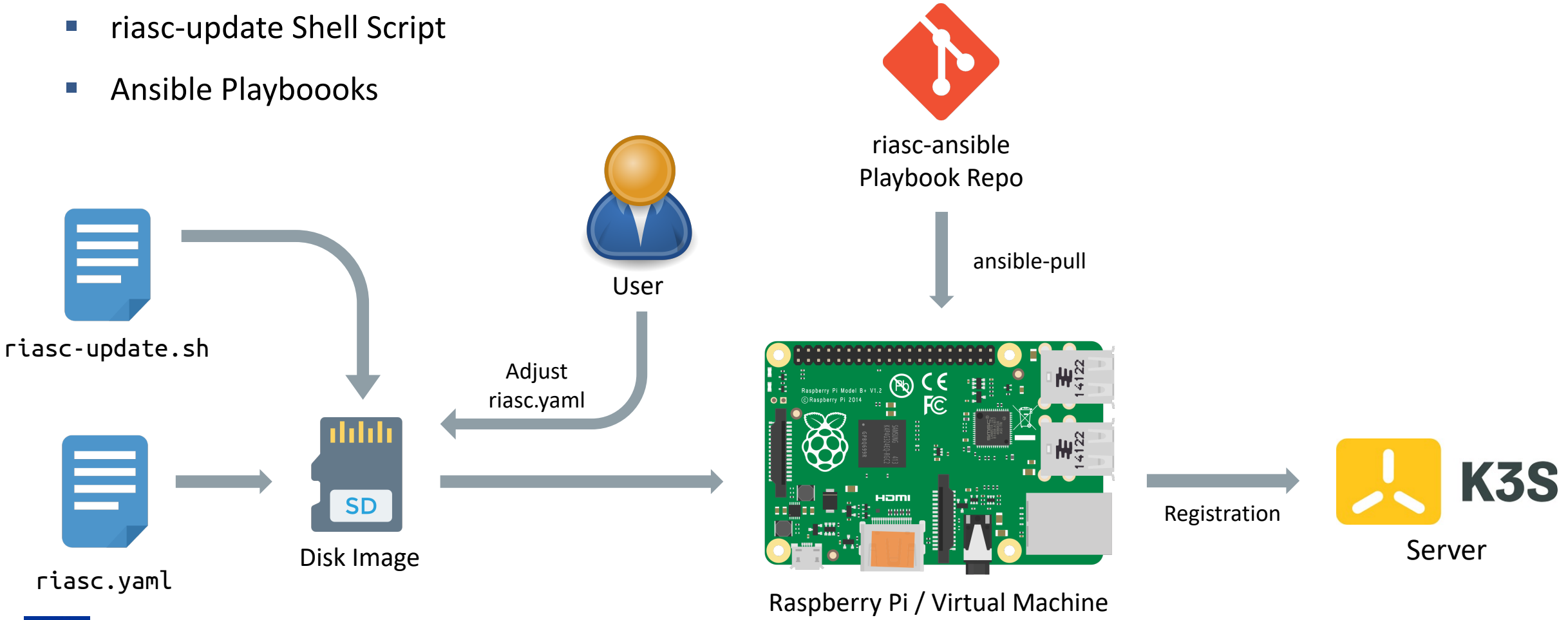


1. Provisioning of Cluster Nodes

- Streamlined process for adding new nodes into the cluster
  - Usually takes 15min to setup a new node
- Requirements:
  - Raspberry Pi or Debian/Redhat-based distro
  - Internet connctivity
  - Hostname and Token
- Receipe
  1. Download image image from riasc.eu
  2. Copy to SDcard
  3. Adjust `riasc.yaml` configuration file on /boot partition
  4. Power-on device and wait…

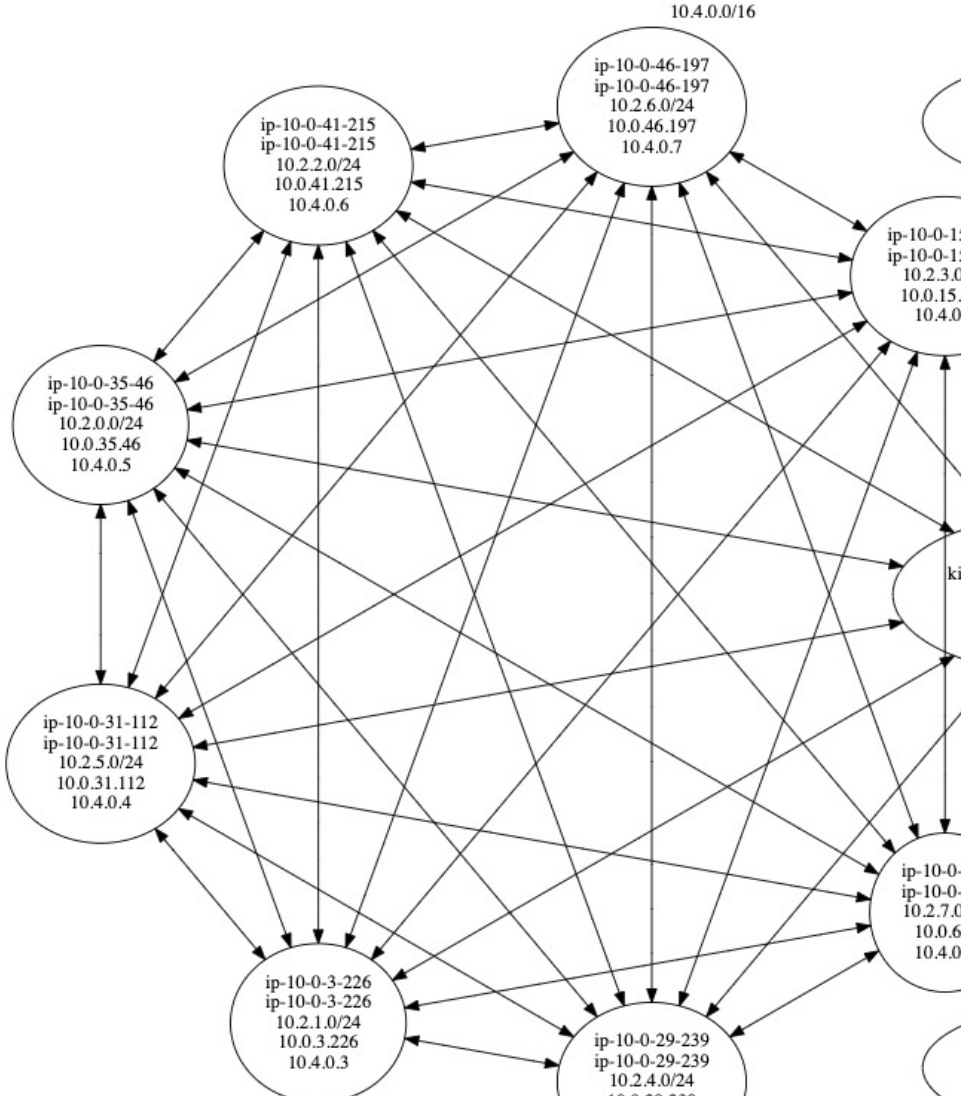
1. Provisioning /2

- Automated SDcard image builder
- riasc-update Shell Script
- Ansible Playboooks



2. Virtual Networking

- All Containers* in the cluster are part of the same cluster internal network
  - Each container* gets a dedicated IP address assigned
- Usually not a problem for centralized clusters
  - However, requires some addons for distributed clusters
- Kubernetes addons: Kilo & Wireguard
- Automatic NAT traversal



2. Network Emulation

- Allows the user to declaratively define network coditions between Pods in the cluster
- Uses Linux Traffic Control subsystem (netem Queuing Discipline)
- Live update of filters without restart of Pods
- Multiple “controllers” available
  - Builtin: simple tc-netem settings
  - Flexe: VTT’s network emulation tool
    - Supports segment sequencing to simulation alternating network conditions

2. Network Emulation /2

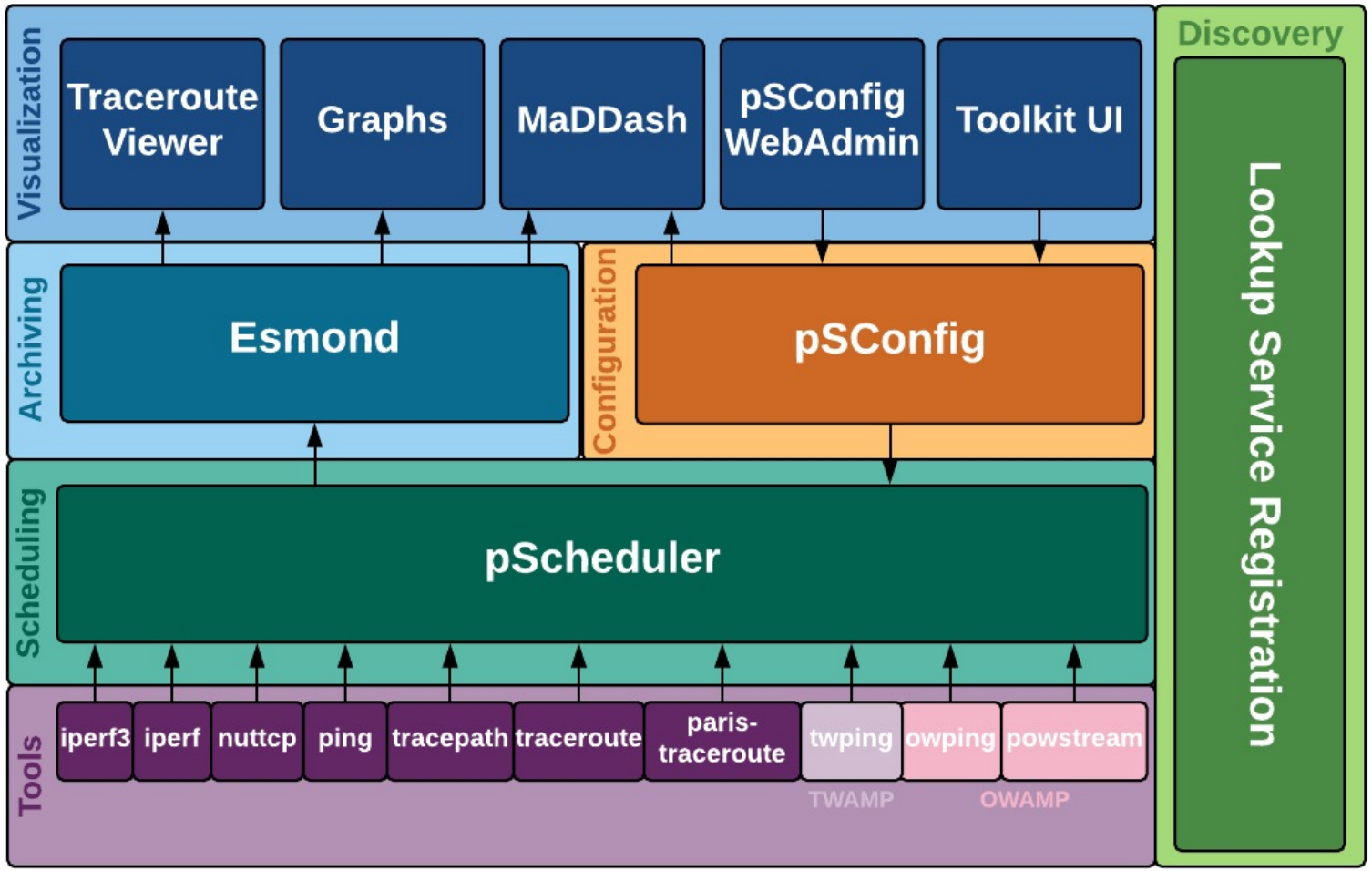
```yaml
---
apiVersion: k8s-netem.riasc.eu/v1
kind: TrafficProfile
metadata:
  name: profile-delay-jitter
spec:
  podSelector:
    matchLabels:
      application: py_ctrl

  type: Builtin
  parameters:
    netem:
      delay: 0.2 # seconds
      jitter: 0.05
      loss_ratio: 1 # in [0, 1]
      distribution: “normal”
```

```yaml
egress:
- to:
  - ipBlock:
      cidr: 1.1.1.1/32
  - podSelector:
      matchLabels:
        component: example

- to:
  - ipBlock:
      cidr: 192.168.0.44/32
  ports:
  - port: 12000
    protocol: UDP
```

2. Network Monitoring

- Work-in-Progress

- Based on perfSONAR
- Operator for configuring test schedules via Kubernetes manifests
- Testpoints run on nodes in the labs
- Central cloud is running web interfaces and measurement archive



2. Network Monitoring /2

```yaml
---
apiVersion: riasc.eu/v1
kind: SchedulerConfig
metadata:
  name: test
spec:
  groups:
    latency_group:
      type: mesh
      addresses:
      - name: rpi-rwth-1
      - name: ntnu-1
  tests:
    latency_test:
      type: latencybg
      spec:
        source: "{% address[0] %}"
        dest: "{% address[1] %}"
        packet-interval: 0.1
        packet-count: 600
```

```yaml
archives:
  esmond_archive:
    archiver: esmond
    data:
      url: https://esmond.riasc-system
      measurement-agent: "{% scheduled_by_address %}"
schedules:
  every_4_hours:
    repeat: PT4H
    slip: PT4H
    sliprand: true
tasks:
  latency_task:
    group: latency_group
    test: latency_test
    archives:
    - esmond_archive
```

3. Time Synchronization

- Runs a dedicated Pod on nodes with
  - Chrony (NTP), GPSd and ptp4linux containers
- Direct hardware access
  - Network cards (PTP), GPS Receivers, PPS Inputs
- Synchronization sources:
  - NTP: Network Time Protocol
  - GPS: Global Positioning System
  - PPS: Pulse per Second trigger
  - PTP: Precision Time Protocol
- Configured via Helm chart
- Can act as NTP server and client
  - Redistribute time within a lab…

EU H2020 Programme GA No. 870620

Time Synchronization /3

```yaml
---
apiVersion: riasc.eu/v1
kind: TimeSyncConfig
metadata:
  name: rwth
spec:
  nodeSelector:
      kubernetes.io/hostname: rpi-rwth-1

  gps:
    enabled: true
    device: ttyAMA0

  pps:
    enabled: true
    device: pps0
    pin: 18
```

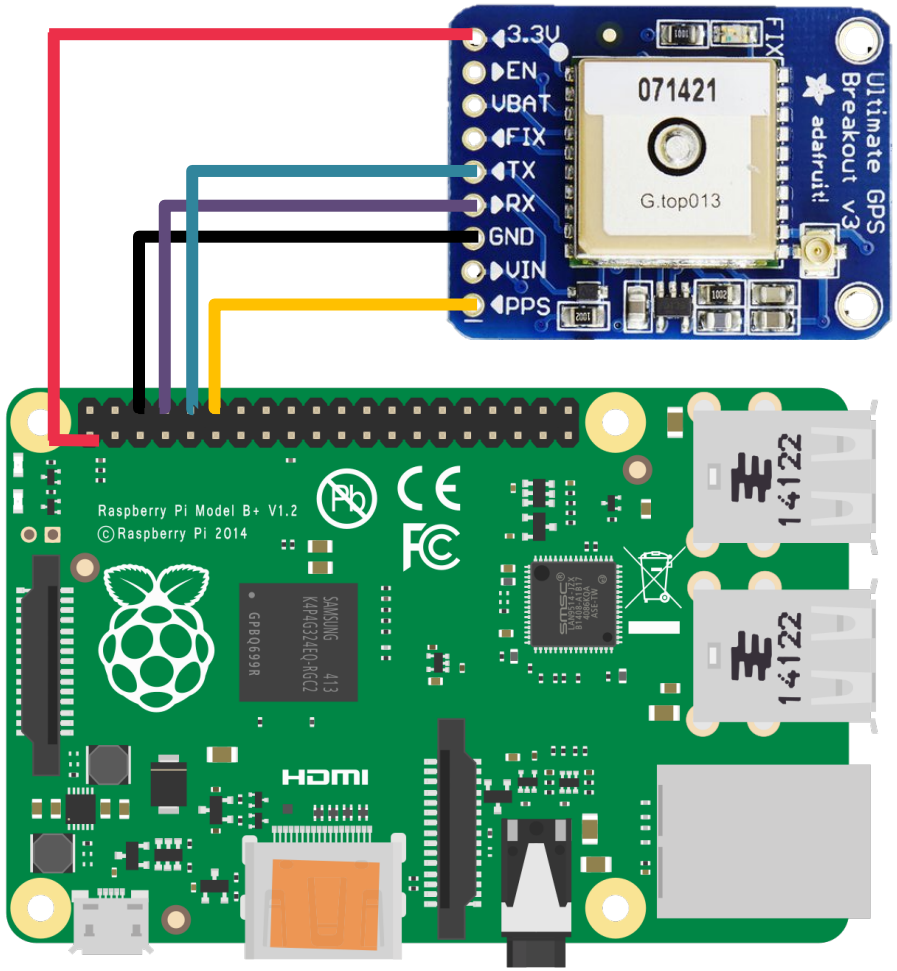
```yaml
ntp:
  server:
    enabled: true
    local: true
    stratum: 1
    allow:
    - 134.130.169.0/25

  servers:
  - address: ntp1.rwth-aachen.de
  - address: ntp.acs-lab
```

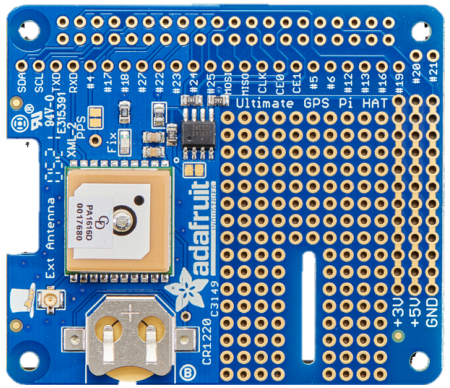

EU H2020 Programme GA No. 870620

Time Synchronization /2

- Low Cost Solution for Raspberry Pi
- Adafruit Ultimate GPS Module v3 (~30 USD)
  - Or Adafruit Ultimate GPS Hat
- Provides Time-of-Day via UART port (NMEA)
- Provides Start-of-Second via Pule-per-Second output (PPS)
- Raspberry Pis have no support for PTP (IEEE1588)

Adafruit Ultimate GPS Module





Adafruit Ultimate GPS Hat

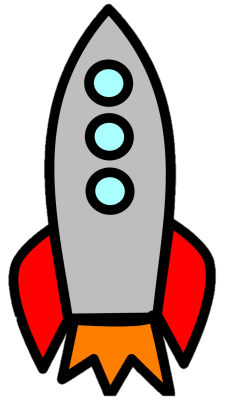
Multi-tenancy

- Allows to restrict access of a user-group to a limited set of nodes
- Implemented as an operator and CRD
- Creates Kubernetes Namespace and Access control lists for us

```
---
apiVersion: riasc.eu/v1
kind: Project
metadata:
  name: sim-hes-off
spec:
  users:
  - steffen-vogel
  - erick-alves

  nodes:
  - rpi-rwth-1
  - ntnu-1
```

Getting Started

1. Provision your own node(s)
   - https://riasc.eu/docs/setup/agent
2. Request user account & project for Cluster access
   - Mail: svogel2@eonerc.rwth-aachen.de
3. Read Quickstart guide
   - https://riasc.eu/docs/usage (WIP)

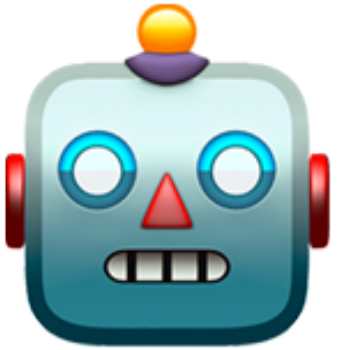


EU H2020 Programme GA No. 870620

Write your own Operator

- kopf: Kubernetes Operator Framework
  - Write operators easily in Python
  - Single file per Operator

- Receipe:
  1. Define ressource description (CRD)
  2. Implement operator logic
  3. Run / Deploy operator
     - Locally: “`kopf run my_operator.py -verbose`”
     - As a container within the cluster itself



Example: Device Configuration

CRD Definition

```yaml
---
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: chromq4qs.device.riasc.eu
spec:
  scope: Cluster
  group: device.riasc.eu
  names:
    kind: Chroma4Q
    plural: chroma4qs
    singular: chroma4q
```

```yaml
  versions:
  - name: v1
    served: true
    storage: true
    schema:
      openAPIV3Schema:
        type: object
        properties:
          spec:
            type: object
            properties:
              state:
                type: string
                default: disconnected
                pattern: '^(dis)?connected$'

                [...]
```


EU H2020 Programme GA No. 870620

Example: Device Configuration

Example Manifest

```yaml
---
apiVersion: device.riasc.eu/v1
kind: Chroma4Q
metadata:
  name: rwth-chroma
  namespace: sim-hes-off
spec:
  connection:
    host: chroma.acs-lab
    port: 2101
    timeout: 20


  state: disconnected
  phases: [1, 2]
  parameters:
    maxCurrent: 16.0 # A
    overcurrentDelay: 0.0 # s
    maxPower: 3500 # VA
    maxFrequency: 55.0 # Hz
    maxVolageAC: 235.0 # V


  setpoints:
    voltageDC:  [  0.0,   0.0]
    voltageRMS: [230.0, 240.0]
    frequency:  [ 50.0,  51.0]
```

Example: Device Configuration

Operator Logic: Creation

```
import kopf

@kopf.on.create('device.riasc.eu', 'v1', 'chroma4qs')
@kopf.on.resume('device.riasc.eu', 'v1', 'chroma4qs')
def create_or_resume(spec: kopf.Spec, memo: kopf.Memo):
    conn = spec.get('connection')
    params = spec.get('parameters')
    setp = spec.get('setpoints')
    if params is None or conn is None:
        raise kopf.PermanentError('incomplete settings')

    memo.amp = chroma.amp4Q(
        conn.get('host'),
        conn.get('port'),
        conn.get('timeout'), False)

    configure(memo.amp, params, setp)
```

Example: Device Configuration

Operator Logic: Update & Deletion

```python
@kopf.on.update('device.riasc.eu', 'v1', 'chroma4qs')
def update(spec: kopf.Spec, memo: kopf.Memo):
    params = spec.get('parameters')
    setp = spec.get('setpoints')
    if params is None or setp is None:
        raise kopf.PermanentError('incomplete settings')


    configure(memo.amp, params, setp)


@kopf.on.delete('device.riasc.eu', 'v1', 'chroma4qs')
def delete(memo: kopf.Memo):
    memo.amp.disconnect_DUT()
```


EU H2020 Programme GA No. 870620

Example: Device Configuration

Operator Logic: Update & Deletion

```python
@kopf.on.update('device.riasc.eu', 'v1', 'chroma4qs')
def update(spec: kopf.Spec, memo: kopf.Memo):
    params = spec.get('parameters')
    setp = spec.get('setpoints')
    if params is None or setp is None:
        raise kopf.PermanentError('incomplete settings')


    configure(memo.amp, params, setp)


@kopf.on.delete('device.riasc.eu', 'v1', 'chroma4qs')
def delete(memo: kopf.Memo):
    memo.amp.disconnect_DUT()
```


EU H2020 Programme GA No. 870620

Example: Device Configuration

Operator Logic: Monitoring

```
@kopf.timer('device.riasc.eu', 'v1', 'chroma4qs', interval=5)
def measurements(spec: kopf.Spec, memo: kopf.Memo):
    phases = spec.get('phases')


    return {
        'frequency':     [memo.amp.meas_frequency(i)       for i in phases],
        'voltageAC':     [memo.amp.meas_voltage_AC(i)      for i in phases],
        'currentDC':     [memo.amp.meas_current_AC(i)      for i in phases],
        'powerReal':     [memo.amp.meas_power_real(i)      for i in phases],
        'powerReactive':[memo.amp.meas_power_reactive(i) for i in phases],
    }
```

Outlook

- More operators
  - VILLASnode
  - perfSonar

- Full experiment configuration with a single file
  - Helm?

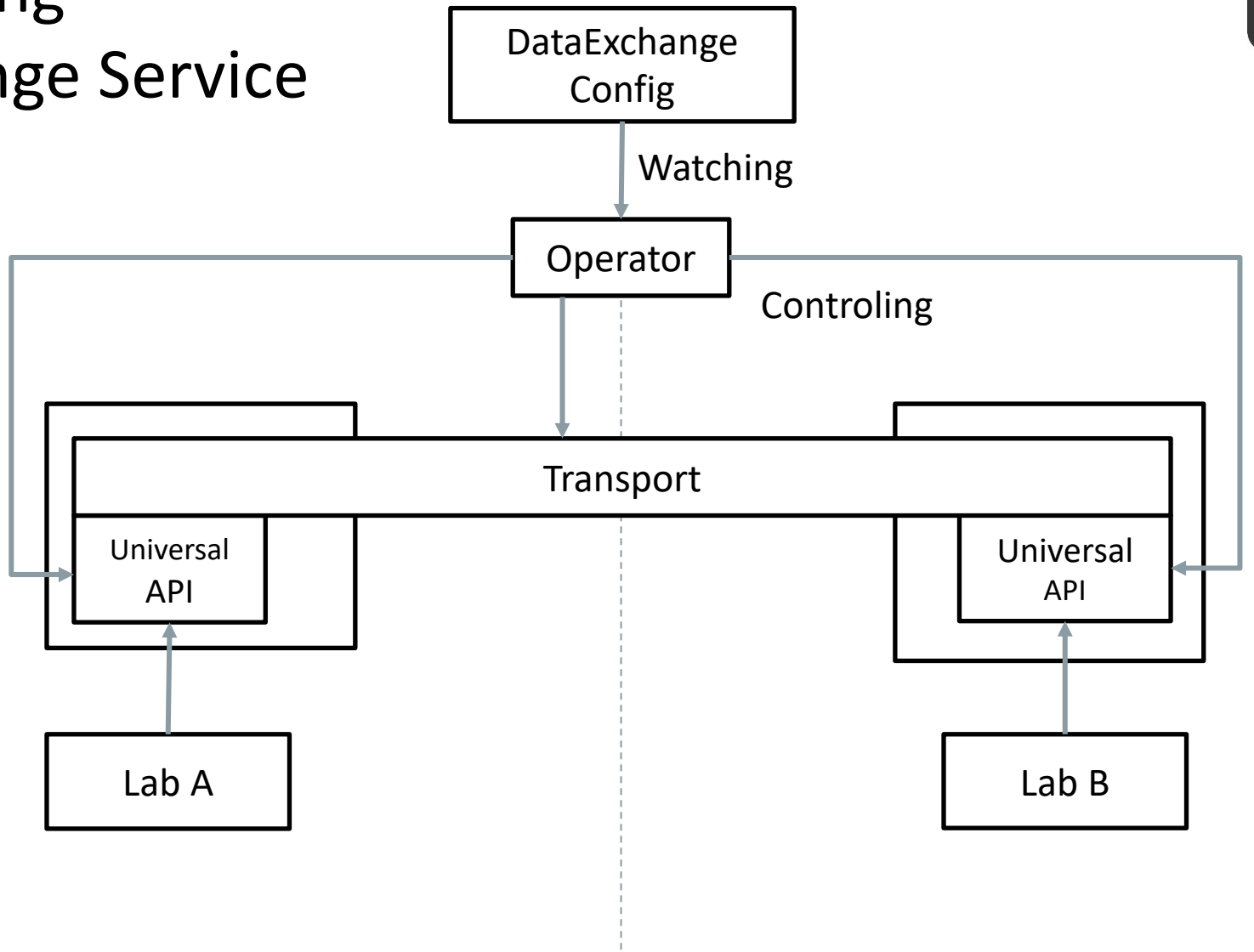
Now: Feedback, Questions & Discussion

Links

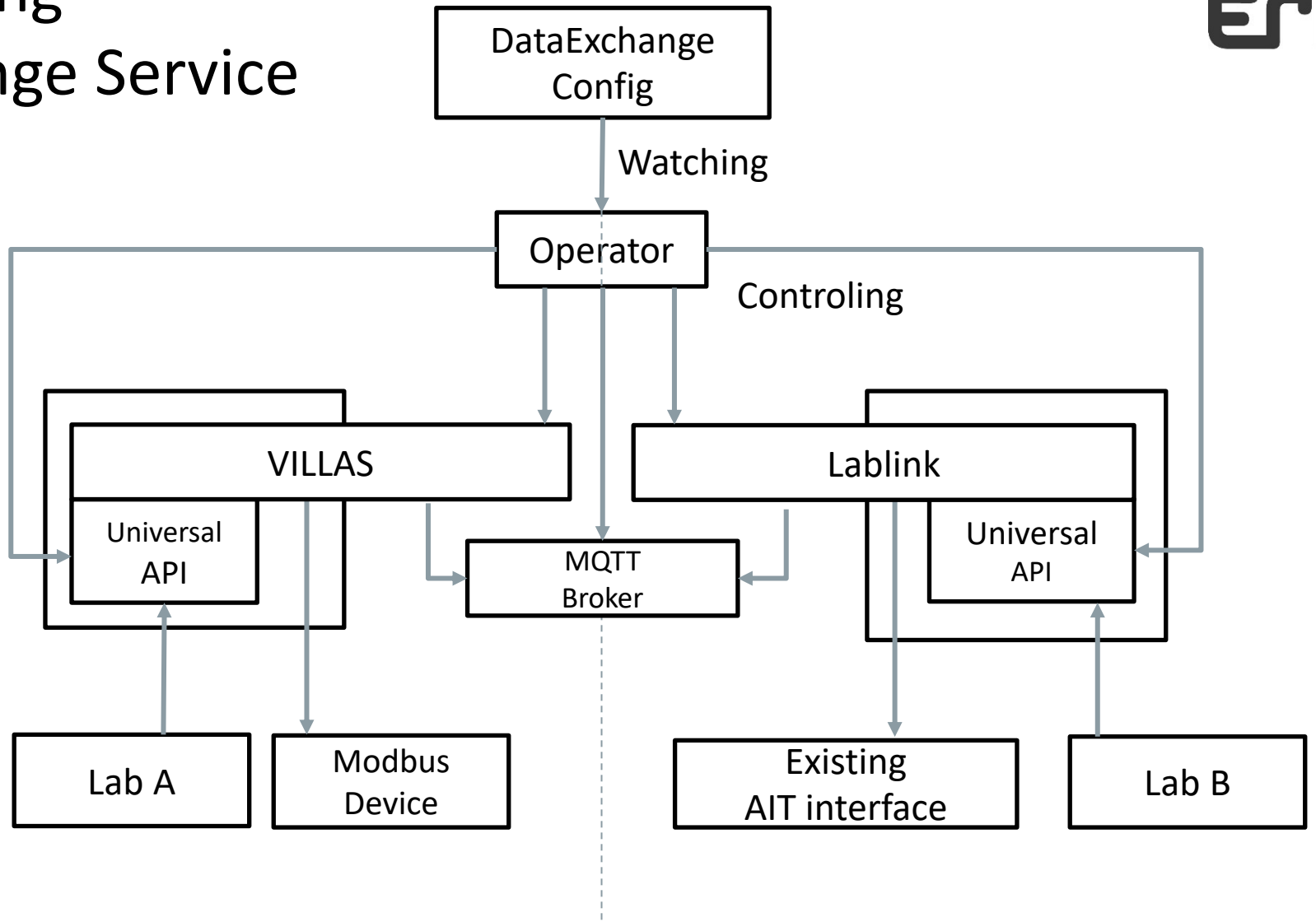
- https://riasc.eu
- https://github.com/orgs/ERIGrid2/teams/riasc/repositories

- https://kubernetes.io

Brainstorming
Data Exchange Service



Brainstorming
Data Exchange Service



Helm