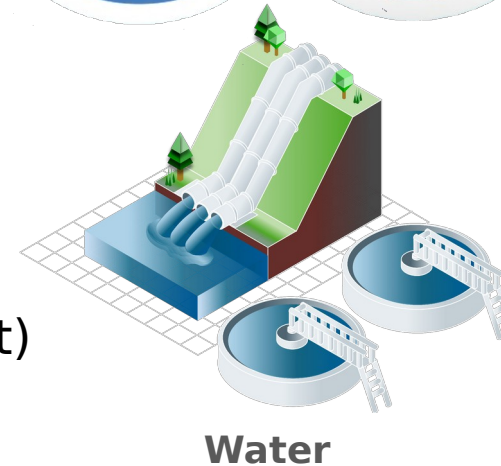


# Cloud-Native Continuum Architectures in the Water Flow Sensing Application Domain

A research to innovation experience report

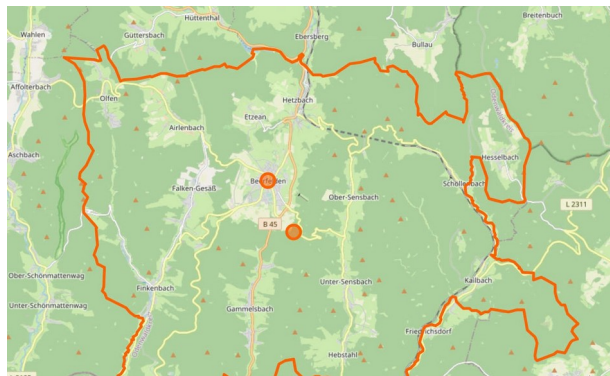


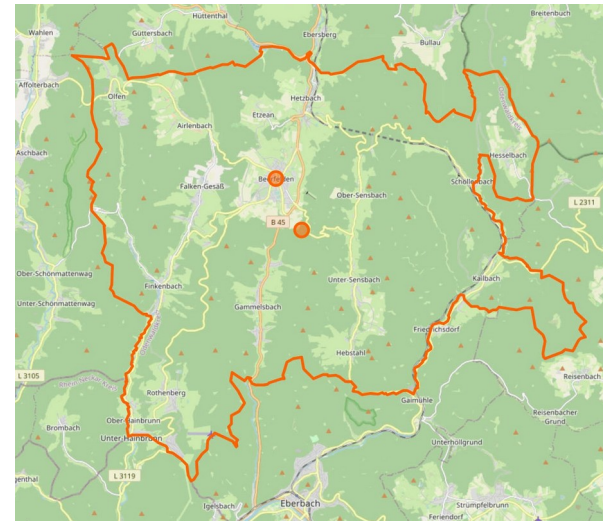
# Application Domain – Distributed Water Flow Sensing



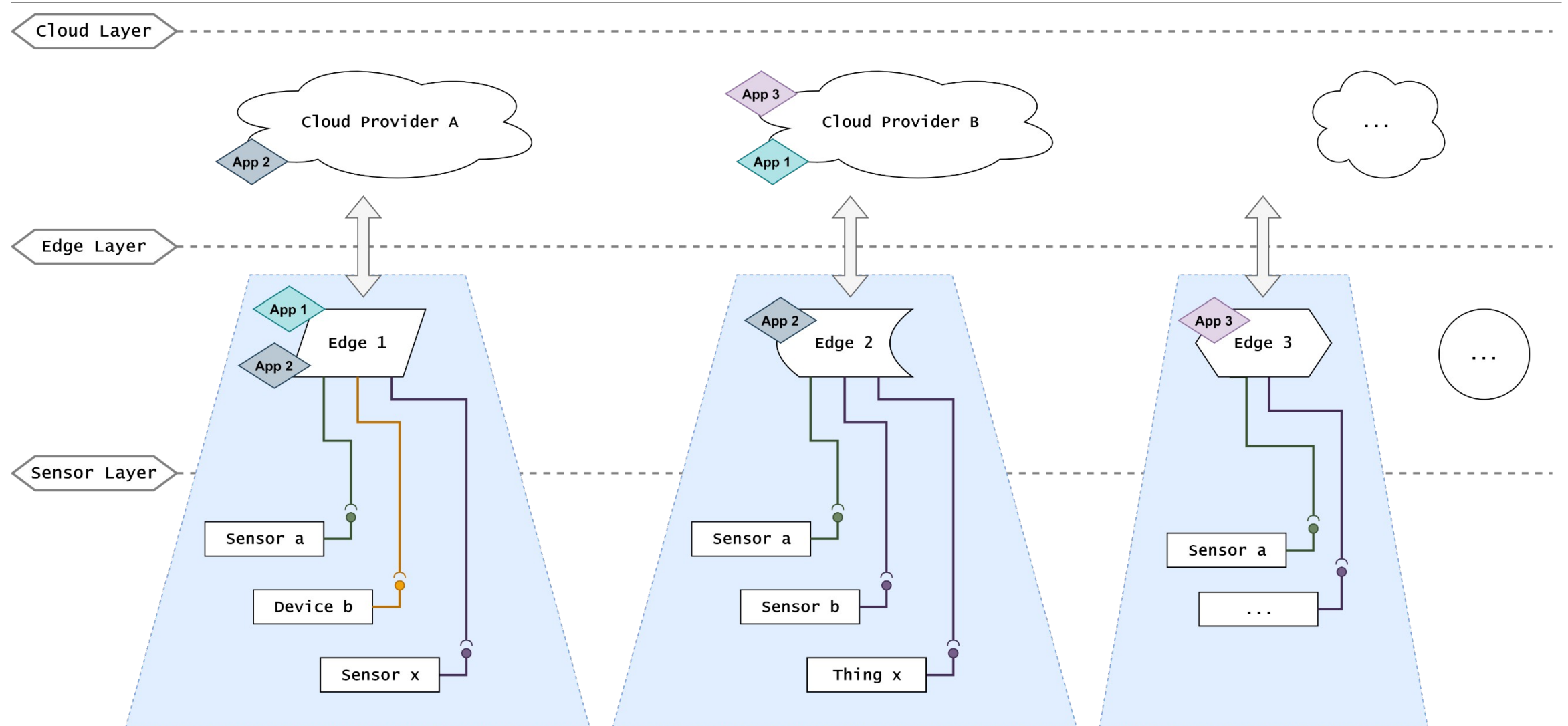
- Benefits through Digitalization
  - Free up capacity of water masters (manual read out)
  - Compliance with legal obligations
  - Optimization potential of operations based on recent data
- Industrial Partner: Endress+Hauser Flowtec
- Academic Partner: Zurich University of Applied Sciences

## Research Challenges

- Development process encapsulation
    - Container-native, same DevOps/GitOps experience for cloud & edge → Kubernetes
    - No push access to edges
    - Functional engineering vs. domain experts
  - Gradual technology selection
    - Fit domain, fit constraints, scale
    - Most effort: throwing away unfit technologies
  - Research approach: Continuum Computing married with IIoT
- pipes (edges) X apps X configs/custom.
- 
- e.g. Oberzent:  
10k people  
60 water sensors @  
10 sites



# The Cloud-to-Things Continuum Model





# Goals and Objectives

---

## Holistic Deployment and Management

- **Allow Deployment and Management of applications across the cloud-to-things continuum**
  - Focus on Cloud and Edge deployment of containerized applications

## Platform Functionality on the Continuum

- **Offer a Cloud-Native developer experience across the cloud-to-things continuum**
  - Allow developers to deploy and maintain applications with a self-service experience
- **Support managed services that are maintained by platform operators**
  - Once per target deployment shall be possible to support resource constraint environments

## Non-functional Objectives

- **Enable faster prototyping and more frequent software releases**

**Research question: do domain-specific application platforms offer merits for developers and platform operators**

## Unfit Existing Technologies (both industry & academic)

- Kubernetes flavours
  - Standalone → no distributed deployment
  - Federation → KubeFed, Admiralty → API endpoint reachable/push-based
  - Extended Cluster → KubeEdge, SuperEdge, OpenYurt, FLEDGE → strong coupl.
- Existing Kubernetes controllers
  - Rancher Fleet → no support for shared services, API endpoint reachable
  - Crossplane → cluster-scoped managed resources (i.e. 1 per cluster)

Criterion	1. Standalone Clusters	2. Cluster Federation	3. Extended Cluster
Target Autonomy	+++	+	+
Communication	+++	+	++
Complexity	+++	++	++
Distributed Deployment Support	+	+++	+++

## Concept

### Holistic Deployment and Management

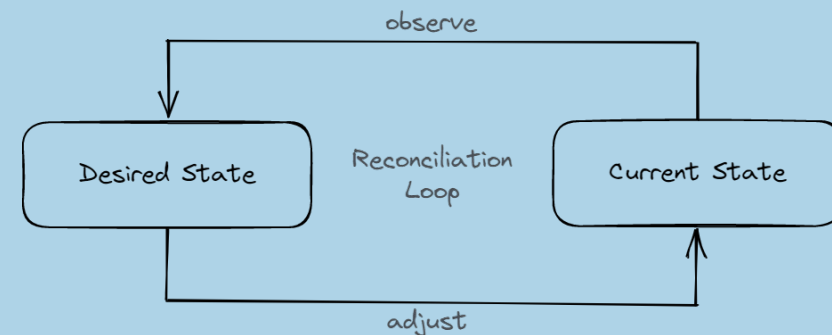
- **Infrastructure Abstraction**
- **Common Deployment Interface**



# kubernetes

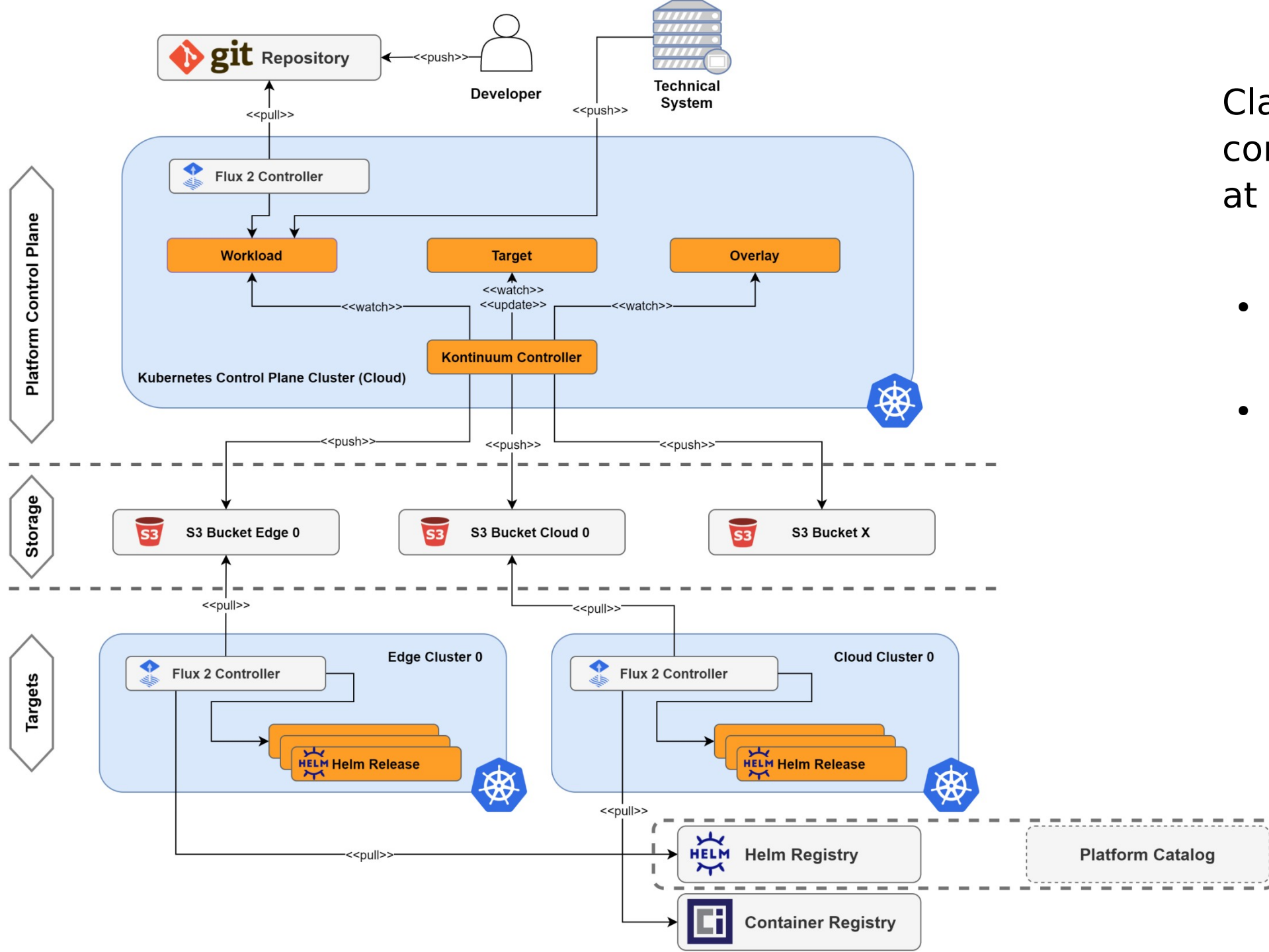
### Platform Functionality on the Continuum

- **Single Deployment Interface for the Continuum**
- **Abstract Kubernetes Complexity**



## Kontinuum Controller

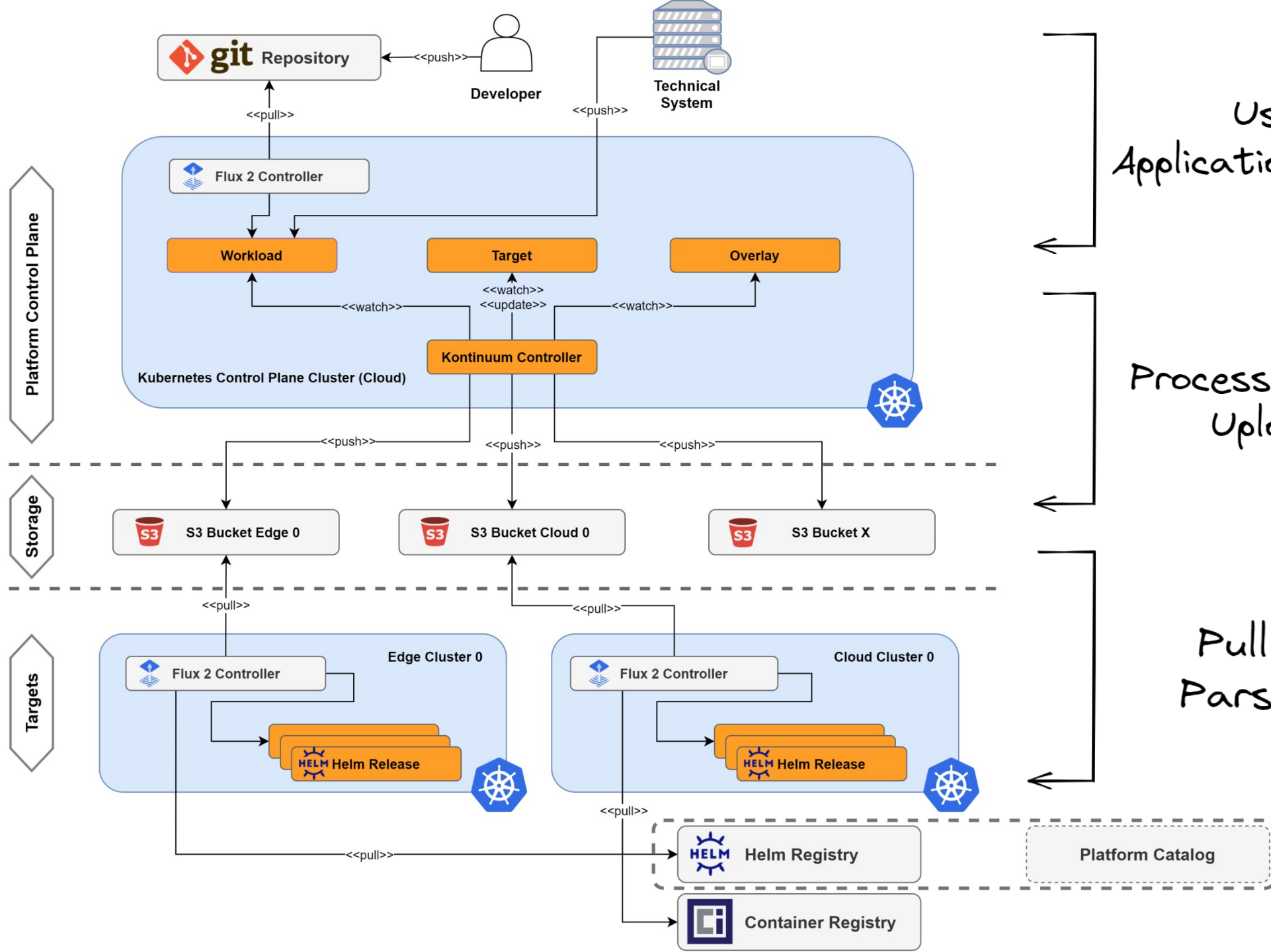
*project name based on the German word "Kontinuum" (engl. continuum)*



Claim: 1<sup>st</sup> K8s-based continuum fit for IIoT at scale

- Distributed deployments
- "Galvanic isolation"



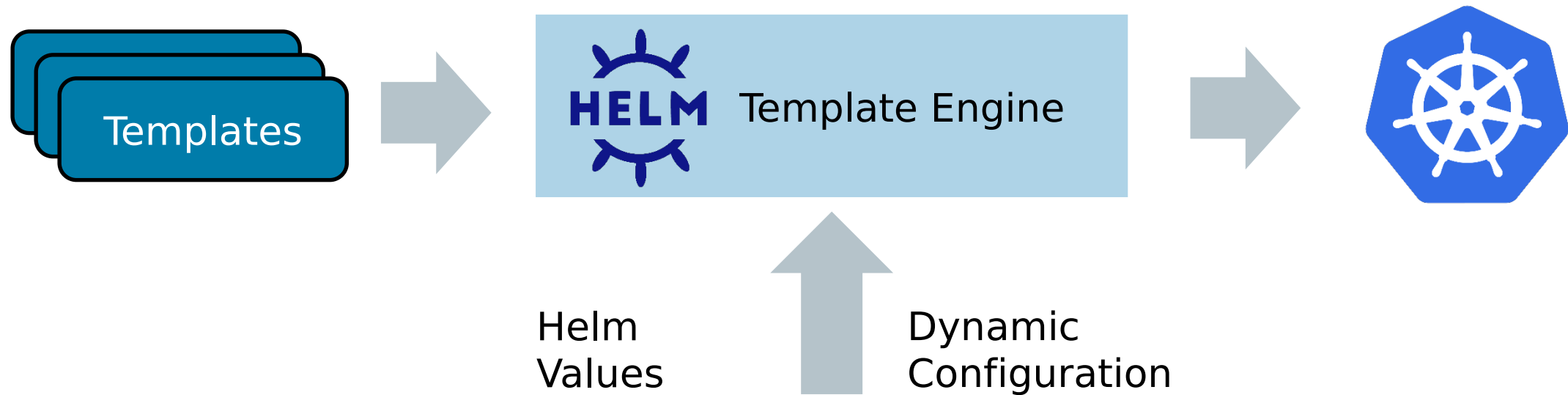


Users define  
Application + Configuration

Process Definitions +  
Upload to S3

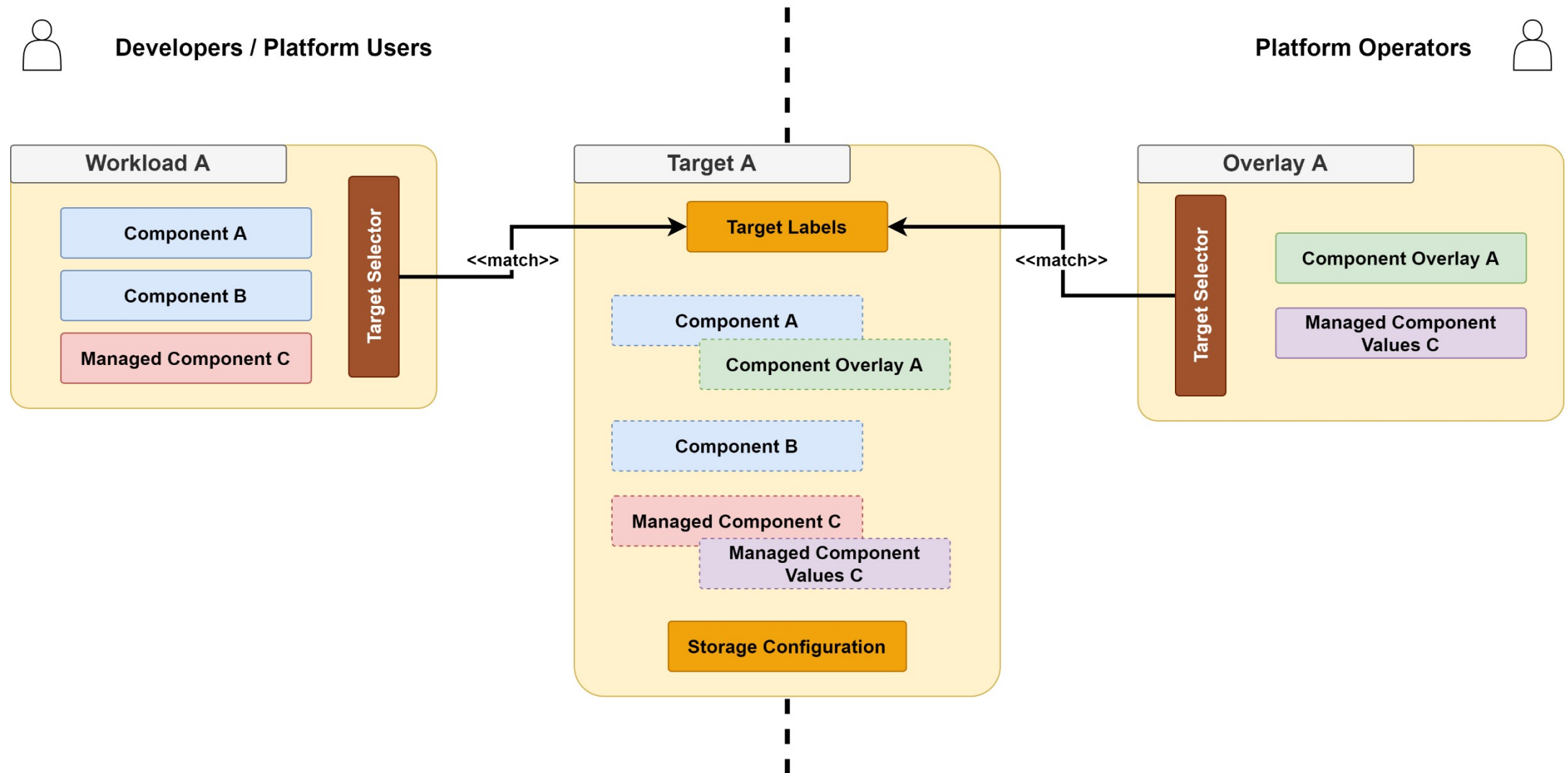
Pull from S3 +  
Parse and Deploy

## Helm Charts + Values as Component Interface




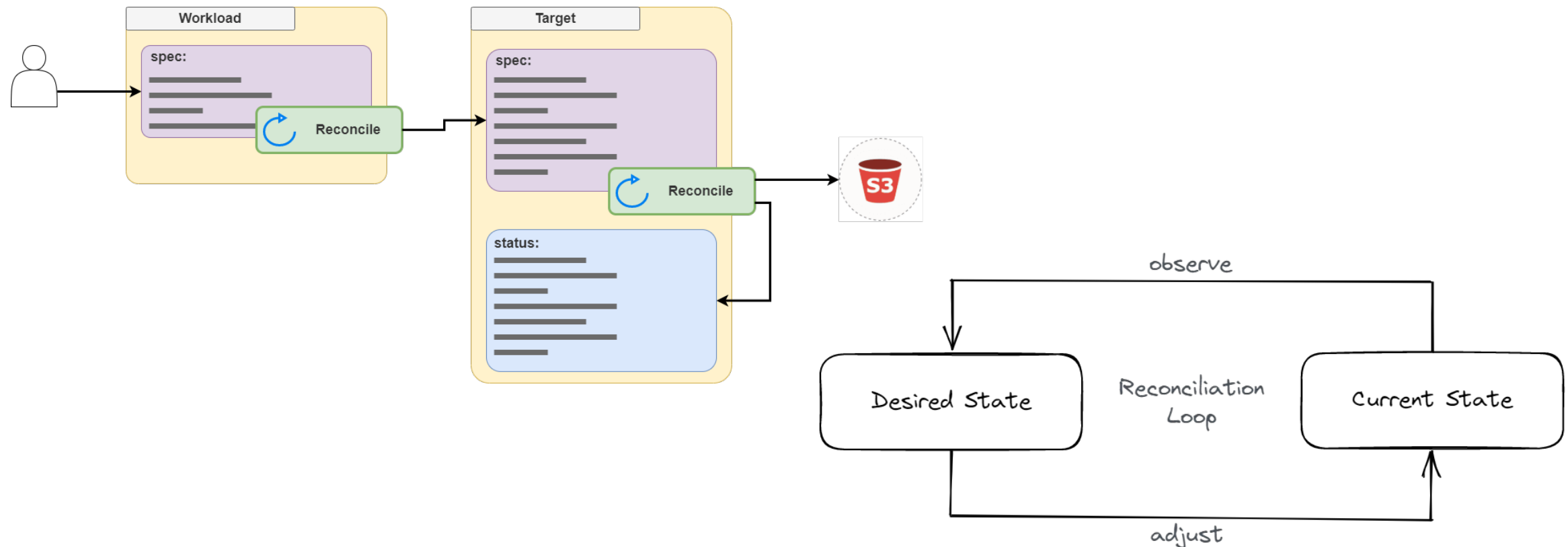
```
1 # Helm Chart: webservice
2 image: example-webservice           # container image to deploy
3 tag: v1.3.4                         # container image tag
4 ingress:
5   domain: webservice.example.com    # domain to publish service externally
6   port: 8080                        # port the container listens on
7 resources:
8   cpu: 1000m                        # CPU limit for the application
9   memory: 128Mi                     # memory limit for the application
```

## Concept – Custom Resource Interface



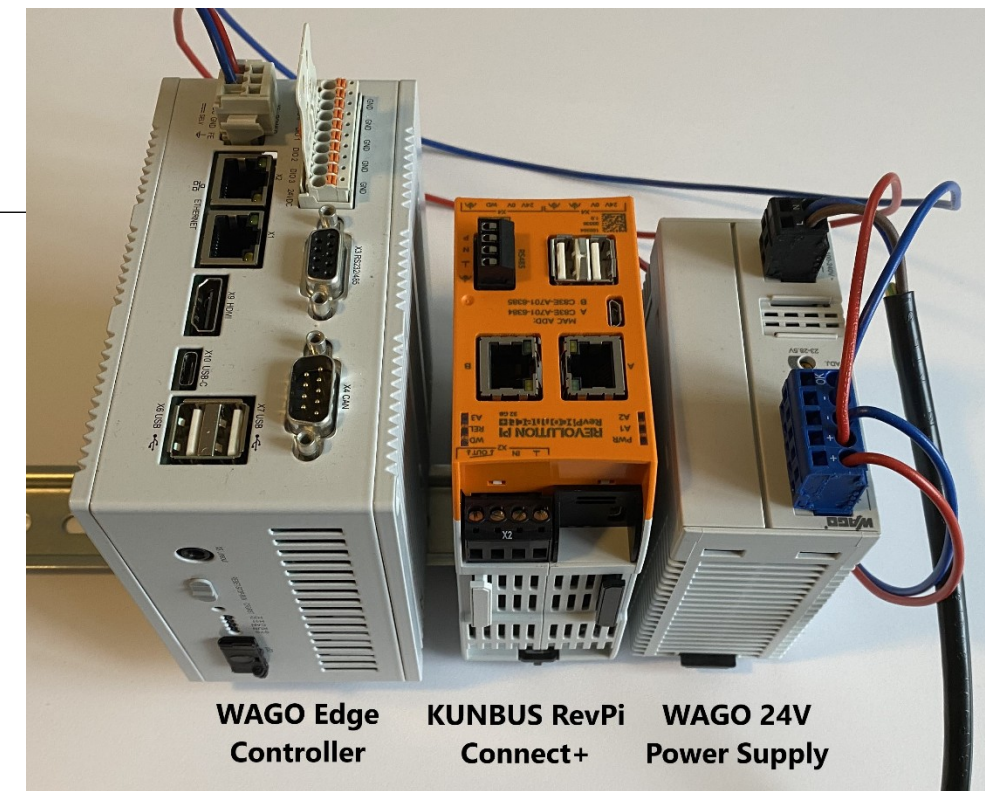
# Kontinuum Controller Internals

- Go based Kubernetes Controller built via the *Kubebuilder* SDK 
- Kubernetes Controller Concept



## Edge Device Resource Consumption

- Testbed Setup
  - Kubernetes Version: k3s v1.21.9
  - Flux Version: v0.24.0



WAGO Edge  
Controller

KUNBUS RevPi  
Connect+

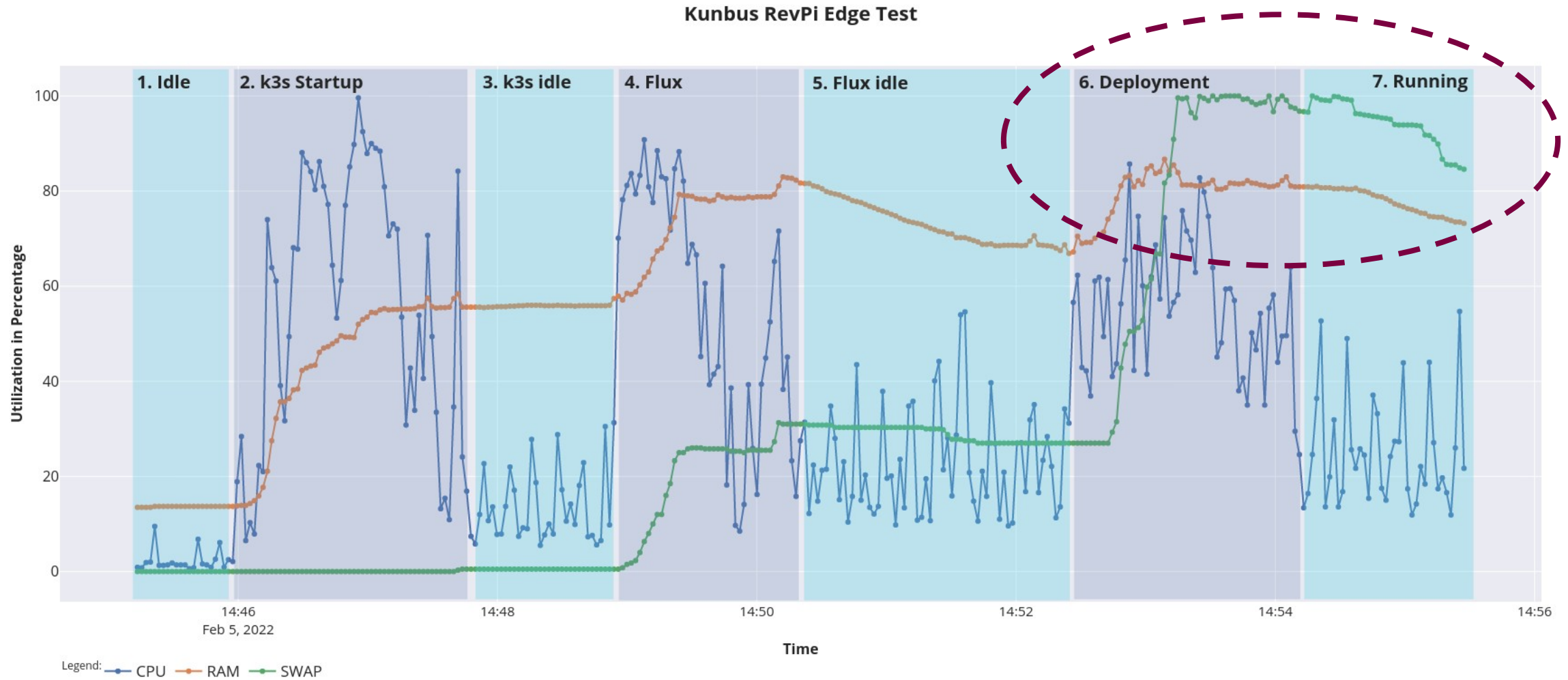
WAGO 24V  
Power Supply

	Kunbus RevPi Connect+	Wago Edge Controller
CPU	4 Core ARM Cortex-A53 @ 1.2 GHz	4 Core ARM Cortex-A9 @ 1.0 GHz
RAM	1 GB	2 GB
Storage	32 GB eMMC	4 GB eMMC
OS	Linux - Raspbian Variant	Linux - Wago Custom
Standards	EN 61131-2 + IEC 61000-6-2	UL 61010-2-201

CPU: ARM A53 @ 1.2 GHz  
RAM: 1 GB



## Results: Kunbus RevPi



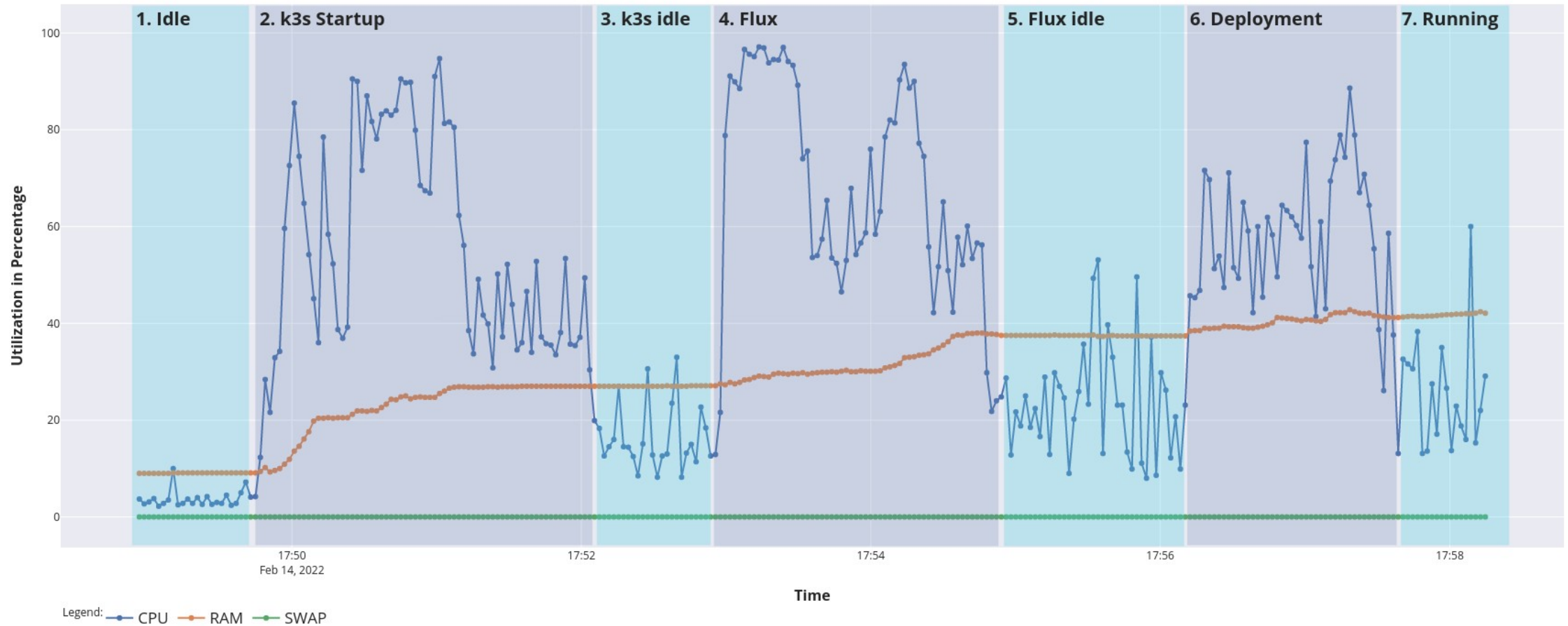


CPU: ARM A9 @ 1.0 GHz  
RAM: 2 GB



## Results: Wago Edge Controller

Wago Edge Controller Test



## Results: Resource Consumption

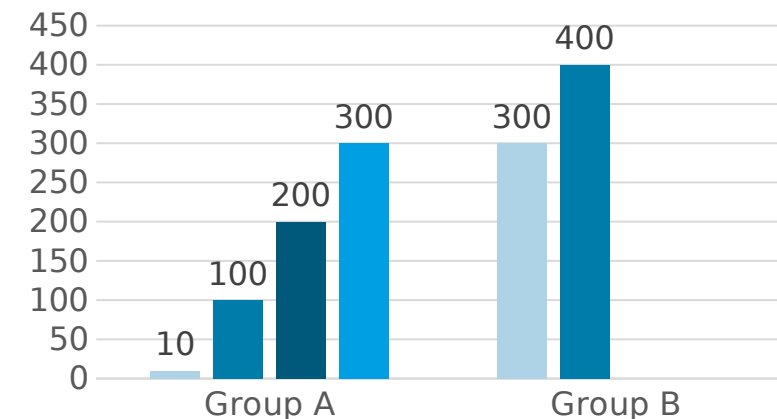
---

- *Kunbus RevPi Connect+*: **1 GB results in clear RAM bottleneck**
- *Wago Edge Controller*: 63% RAM available for applications
  - **~800 MB consumed by platform components**
- CPU consumption in both cases acceptable
  - Idle: **23% of CPU resources consumed by platform components**
  - Upgrades and Startup can be performed during maintenance period
  - Scale to zero for Flux CD-Agent components possible

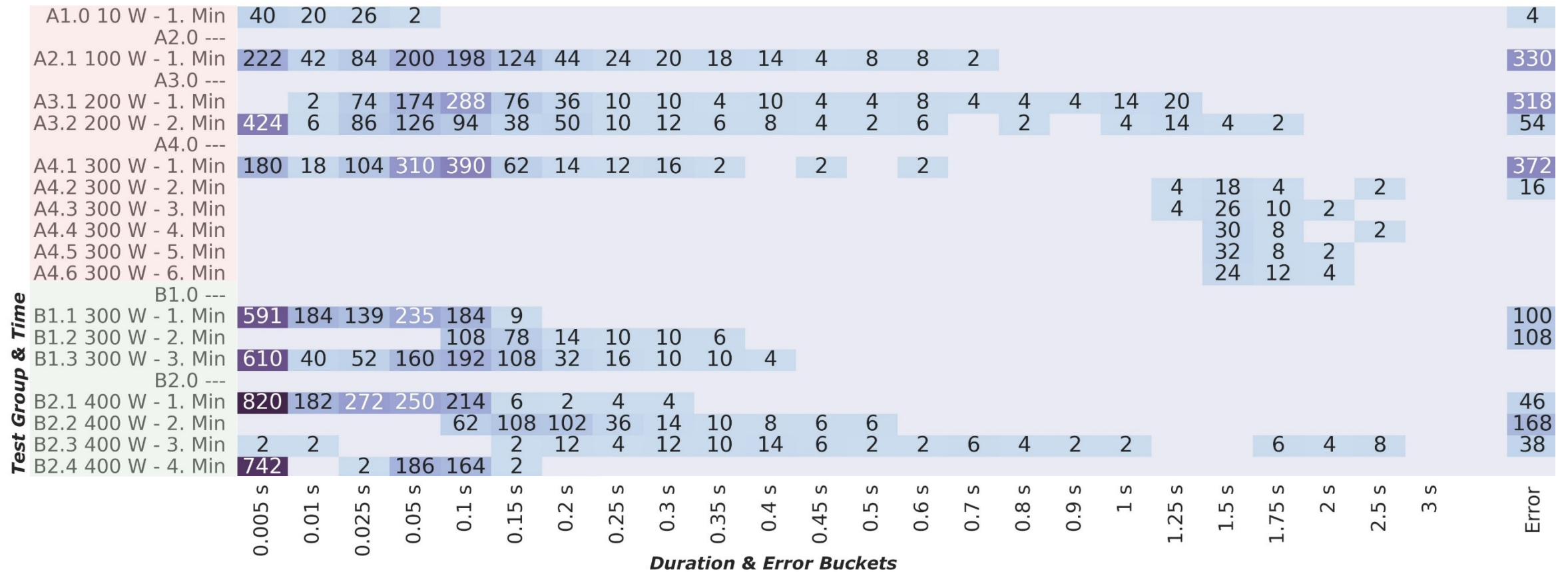
## Kontinuum Controller Scalability

- Test variable: number of workloads per target
  - A large number of workloads per target increases reconcile durations
  - This might cause scalability issues
  - A large number of targets with a small number of workloads can be scaled through parallelism
- Test cases:
  - All tests performed with 10 Overlays and a single Target
  - Group A – original version of the Kontinuum Controller:
    - **A1** 10 Workloads / **A2** 100 Workloads / **A3** 200 Workloads / **A4** 300 Workloads
  - Group B – skipping reconciles for changes to the status section:
    - **B1** 300 Workloads / **B2** 400 Workloads
    - [...]

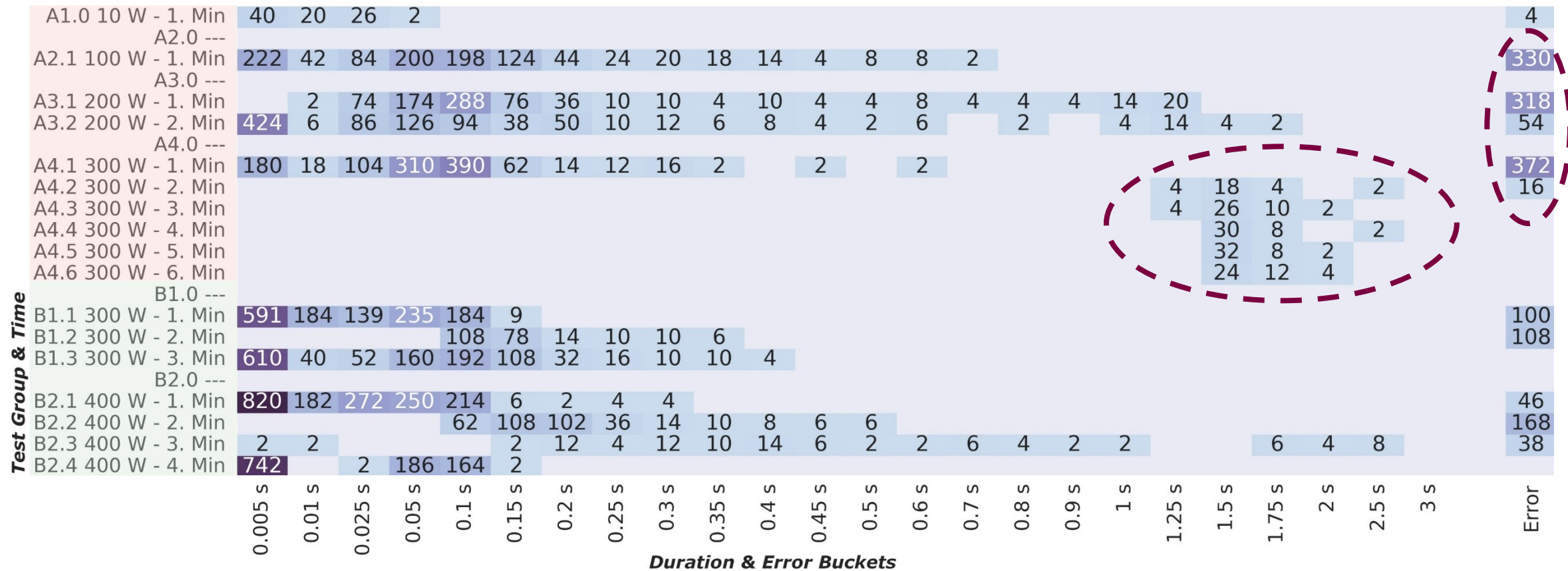
Test Cases Overview



# Kontinuum Controller Scalability

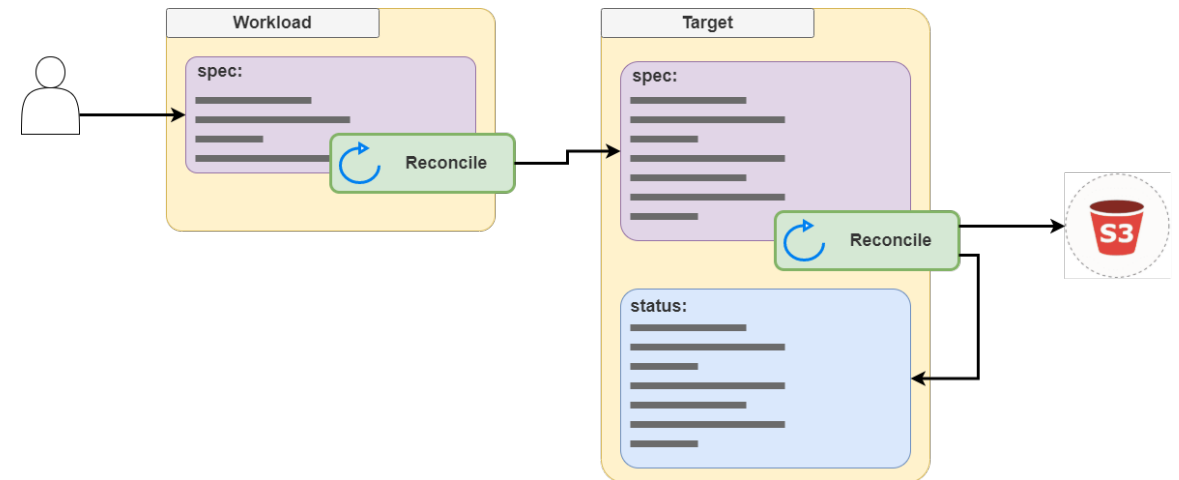


# Kontinuum Controller Scalability



# Kontinuum Controller Scalability

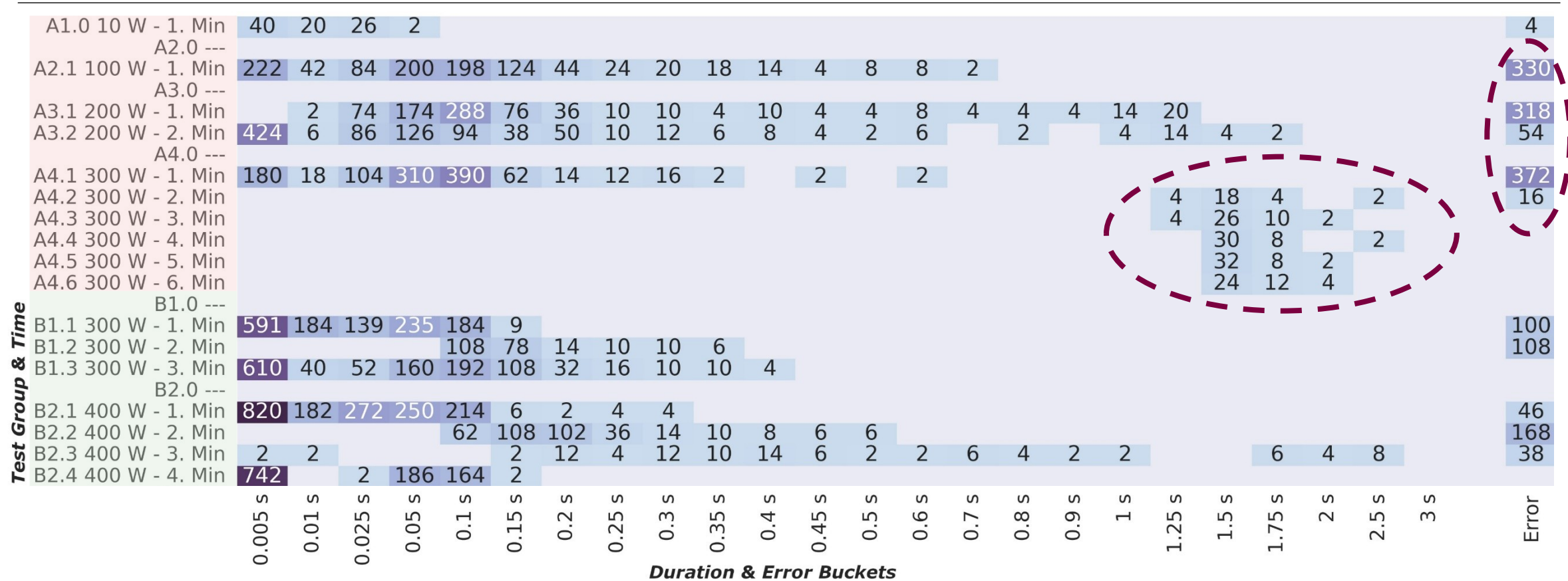
```
apiVersion: crd.kontinuum-controller.github.io/v1alpha1
kind: Target
▼ metadata:
  ► annotations:↔
  ► creationTimestamp: "2022-04-23T11:25:23Z"
  ► generation: 18
  ▼ labels:
    arch: x86
    group: cloud
    name: apu-target
  ► managedFields:↔
  name: apu-target
  namespace: default
  resourceVersion: "1349912"
  uid: 318cdd6a-3b05-4893-9795-39526967efe3
  ► spec:↔
  ▼ status:
    lastUploadTimestamp: "2022-04-27T13:07:33Z"
    ► managedWorkloads:↔
    ► workloads:↔
```



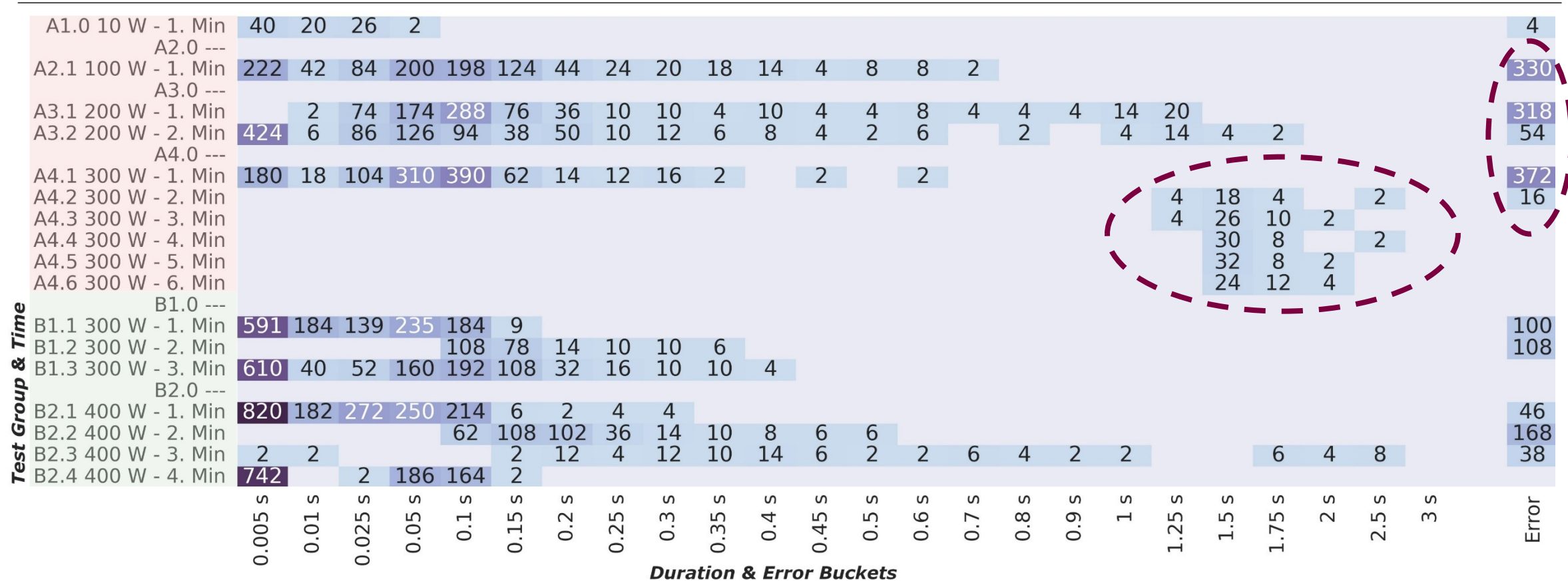
```
1 Operation cannot be fulfilled on targets.kontinuum-controller.github.io
2 "target-eval-0-0": the object has been modified;
3 please apply your changes to the latest version and try again
```



# Kontinuum Controller Scalability



# Kontinuum Controller Scalability



- Promising feature for further improvements: *Server-Side-Apply* (GA Kubernetes v1.22)
  - not yet implemented in Kubebuilder SDK
  - technical PoC using kubectl looks promising

## Reflection & Outlook

---

- The Kontinuum Controller could show its ability to provide holistic deployment and management of applications across the cloud-to-things continuum
- The controller software needs further iterations to overcome the current PoC state
- The concept validation didn't reveal fundamental flaws
  - Experts suggested some minor implementation improvements
  - Hardware for deployment targets should be sized accordingly
- Besides the focus on the water flow sensing application domain
  - Concept should deliver merits to other industries as well
- Thesis can be seen as a first step towards Continuum-Native application platforms

# Questions?

*Thank you for the attention!*

<https://kontinuum-controller.github.io/>

