



Swisscom Managed K8s Hands-on with Camptocamp

Thursday, 4
December 2025



KCD Suisse Romande
DECEMBER 4 & 5, 2025 – CERN, GENEVA

camptocamp



Federico Sismondi

Infrastructure developer



- Platform Developer and Operator
- I really like Kubernetes
- I work hosting Odoo and Geospatial apps
- I garden and grow pumpkins in my free time



Julien Acroute

Infrastructure developer

- Manage Kubernetes Platform
- Automate Deployment
- Passionate about Observability
- Trainer for K8s, Docker, Terraform, PostgreSQL
- PostgreSQL/PostGIS enthusiast



Workshop Agenda



00 Warm up – OpenMetrics

01 Bootstrap – Create the Kubernetes Cluster

02 Init – Install tools

03 Story – Python based Particle Accelerator

04 Run – Deploy the lab and play with the metrics

05 Discover – Observability tools using EBPF

00 – OpenMetrics

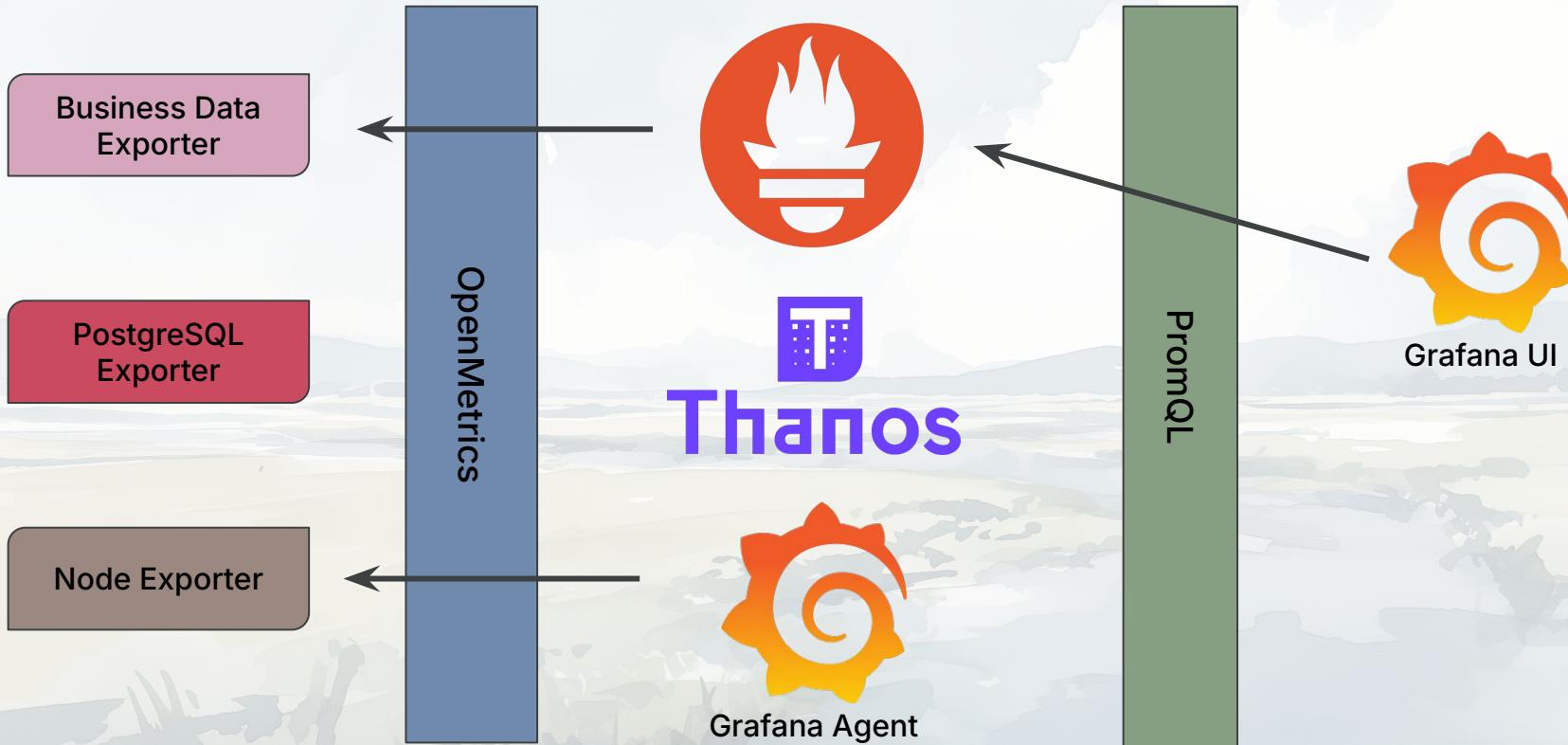


- A standard to expose numeric metrics
- Defined on openmetrics.io
- Extends Prometheus format
- Text based or Protocol Buffers
- CNCF Graduated project since 2018



OPEN METRICS

00 – OpenMetrics



00 – OpenMetrics



Exporter



Prometheus

```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
disk_usage_percent{partition="/"}
```

partition= "/"	63.7
partition= "/var"	37.2
partition= "/tmp"	12.5

00 – OpenMetrics



Labels

- Label: metrics dimension
- A set of key/value pair
- Uniqueness: metric + labels + timestamp
- Shared between metrics

```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
disk_usage_percent{partition="/" 63.7
disk_usage_percent{partition="/var"} 37.2
disk_usage_percent{partition="/tmp"} 12.5
```

```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
disk_usage_percent{partition="/" 63.4
disk_usage_percent{partition="/var"} 37.6
disk_usage_percent{partition="/tmp"} 12.3

# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="GET", code="200"} 1234027
http_requests_total{method="POST", code="200"} 1027
http_requests_total{method="POST", code="400"} 3

# HELP sales_total The total number of sales.
# TYPE sales_total counter
sales_total{category="Tools", brand="Makita"} 163376
sales_total{category="Tools", brand="Bosh"} 298425
sales_total{category="Garden and Outdoor", brand="Weber"} 18346
sales_total{category="Garden and Outdoor", brand="Hyundai"} 163376

# HELP stock The number of stock items.
# TYPE stock gauge
stock{category="Tools", brand="Makita"} 234
stock{category="Tools", brand="Bosh"} 456
stock{category="Garden and Outdoor", brand="Weber"} 27
stock{category="Garden and Outdoor", brand="Hyundai"} 45
```

00 – OpenMetrics



Metrics Type

- HELP Description of the metrics
- TYPE Metrics type:
 - Gauge: can increase and decrease
 - Counter: only increase, cumulative
 - Histogram, Summary

```
# HELP disk_usage_percent Usage of disk in percent (0-100)
# TYPE disk_usage_percent gauge
disk_usage_percent{partition="/"}
```

partition	usage (%)
/	63.7
/var	37.2
/tmp	12.5

00 – OpenMetrics

Custom implementation



```
1 view_metric = {}
2
3 @app.route('/view/<id>')
4 def view_product(id):
5     # Update metric
6     if product not in view_metric:
7         view_metric[id] = 1
8     else:
9         view_metric[id] += 1
10    return "View %s" % id
```

```
1 @app.route('/metrics')
2 def metrics():
3     metrics = ""
4     for id in view_metric:
5         metrics += 'view_total{product="%s"} %s\n' %
6                     (id, view_metric[id])
7     for id in buy_metric:
8         metrics += 'buy_total{product="%s"} %s\n' %
9                     (id, buy_metric[id])
10    return metrics
```

```
view_total{product="p1"} 17
view_total{product="p2"} 25
buy_total{product="p1"} 5
buy_total{product="p2"} 12
```

00 – OpenMetrics

Prometheus Libraries



- Custom implementation
- Libraries: Python, Go, Java, Ruby, Rust, ...

```
1 from prometheus_client import Counter
2
3 view_metric = Counter('view', 'Product view', ['product'])
4
5 @app.route('/view/<id>')
6 def view_product(id):
7     # Update metric
8     view_metric.labels(product=id).inc()
9     return "View %s" % id
```

```
1 from prometheus_client import generate_latest
2
3 @app.route('/metrics')
4 def metrics():
5     return generate_latest()
```

00 – OpenMetrics

Helpers



- Exception counter
- Track duration

```
1 import time
2 import random
3 from prometheus_client import Summary
4
5 duration = Summary('duration_compute_seconds', 'Time spent in compute()')
6
7 @duration.time()
8 def compute():
9     time.sleep(random.uniform(0, 10))
```

00 – OpenMetrics

On-demand Metrics



- Use callback

```
1 from prometheus_client import Counter
2
3 stock_metric = Counter('stock', 'Stock count')
4 stock_metric.set_function(compute_stock)
5
6 def compute_stock():
7     res = query('SELECT count(*) FROM product_stock')
8     for line in res:
9         return line[0]
```



01 – Create the Kubernetes Cluster

Get the Welcome card

The welcome card contains:

- URL to the Swisscom Web UI: <https://xxxx.ks.private.cloud.swisscom.ch/>
- Username
- Password

The welcome page contains:

- Lab Instructions
- Commands for copy/paste
- <https://lab.campto.camp/>

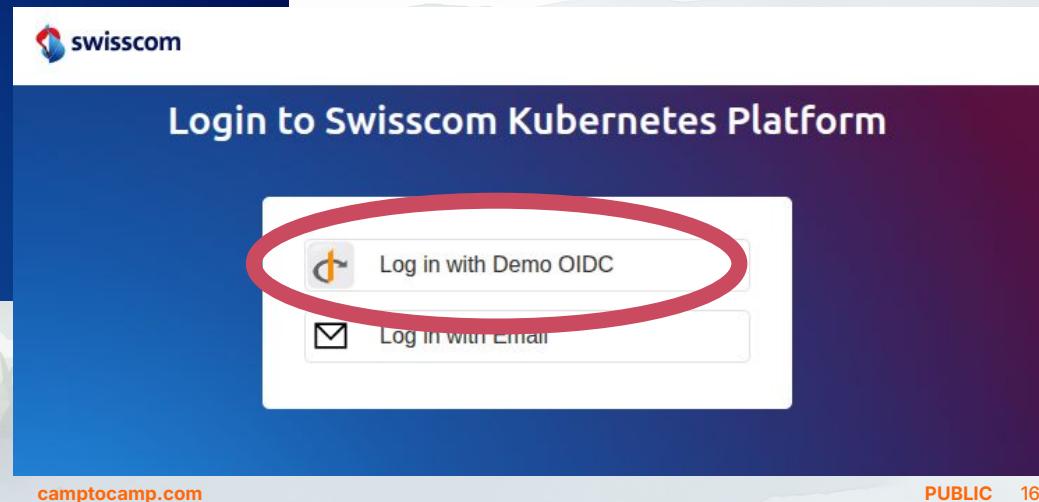
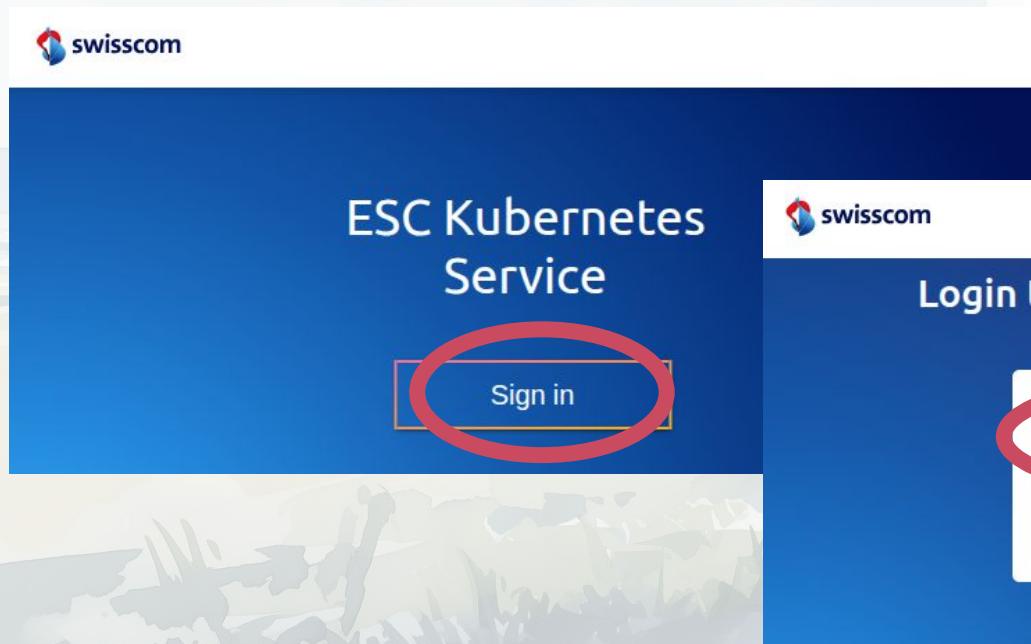




01 – Create the Kubernetes Cluster

Sign in

Using URL from the welcome card: <https://xxxx.ks.private.cloud.swisscom.ch/>





01 – Create the Kubernetes Cluster

Choose project

The screenshot shows the Swisscom Cloud Control Platform's 'Projects' page. At the top, there is a search bar labeled 'Search'. Below it, a list of projects is displayed. The first project, 'camptocamp-01', is highlighted with a green dot icon. It includes details such as its ID (partially obscured), a CPU icon with the value '1', a memory icon with the value '0', a CI/CD icon with '+1', and an owner icon. Below these details, there are three progress bars for CPU, Memory, and Disk resources.

Resource	Value
CPU	1
Memory	0
Disk	0



01 – Create the Kubernetes Cluster

Create a new cluster

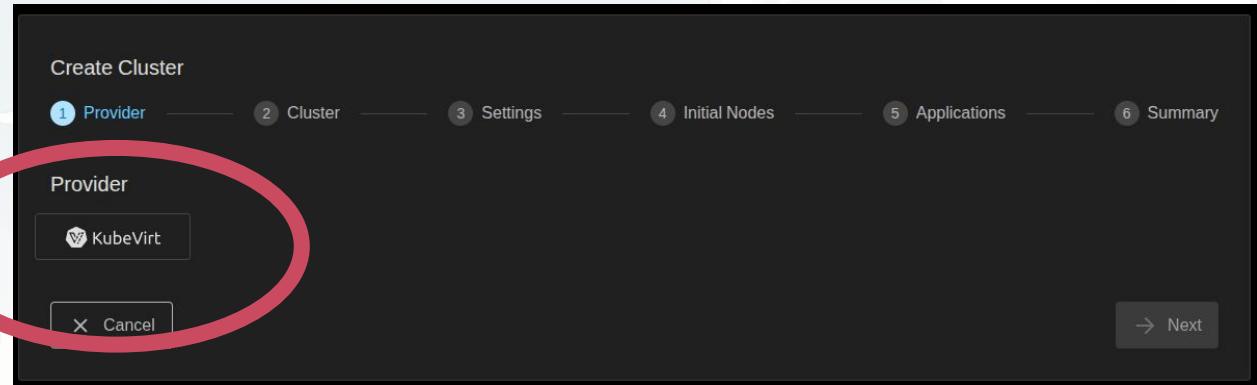
- Create Resource
 - Cluster

The screenshot shows the Camptocamp web interface with a dark theme. At the top, it displays the project name "campocamp-02". The main area is titled "Project Overview" and contains sections for "Project", "Clusters", "External Clusters", "Cluster Templates", "Backups", and "SS". On the left, there is a sidebar with various icons. A red circle highlights the "Create Resource" button in the top right corner of the main content area. Below the main content, there are two more sections: "Project Quota" and "Cluster Distribution", both indicating "No clusters available." A blue button labeled "Show All Clusters" is visible in the "Clusters" section.



01 – Create the Kubernetes Cluster Provider

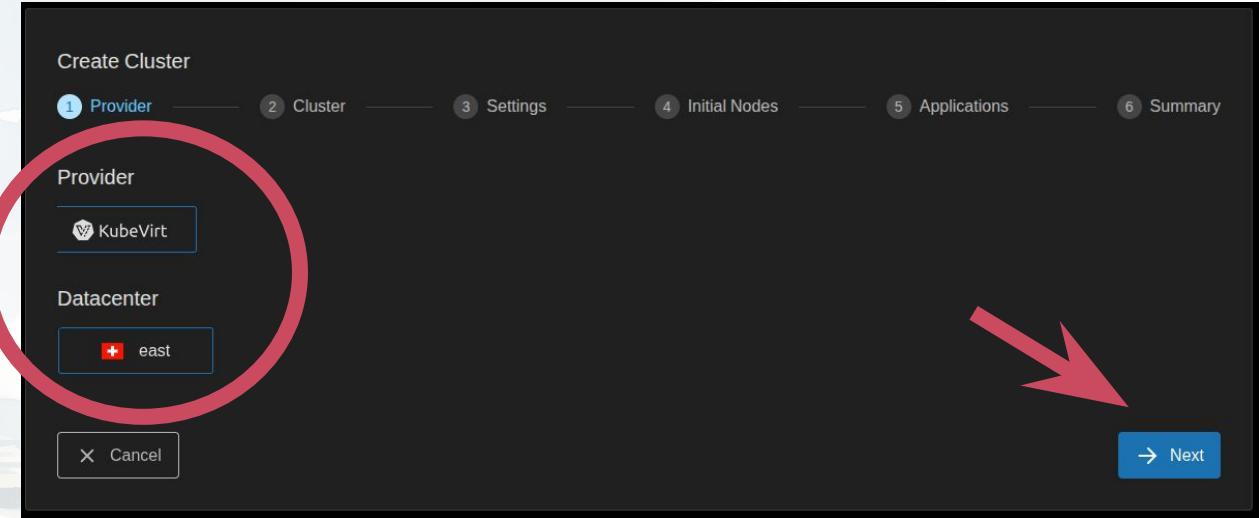
- Select Provider





01 – Create the Kubernetes Cluster Provider

- Select Provider
- Select Datacenter



01 – Create the Kubernetes Cluster

Cluster



- Cluster name

Create Cluster

Provider ✓ Cluster ✓ Settings ✓ Initial Nodes ✓ Applications ✓ Summary ✓

Cluster

Name*

Network Configuration

cilium canal None

CNI Plugin Version

Specification

Control Plane Version* Container Runtime*

Admission Plugins

Audit Logging custom metadata minimal recommended (i)

Cluster Backup (i)

Disable CSI Driver (i)

Annotations

Key Value

→ Next ← Back



01 – Create the Kubernetes Cluster

Settings

- Select settings

Create Cluster

Provider Cluster Settings Initial Nodes Applications Summary

Basic Settings

Provider Preset* swisscom-east

Kubeconfig*

No VPCs Available

Please enter your credentials first.

Cancel Back Next

camptocamp.com



01 – Create the Kubernetes Cluster

Initial Nodes

- Deploy only one node

Create Cluster

Provider Cluster Settings **Initial Nodes** Applications Summary

Basic Settings

Leave this field blank to use automatic name generation.
Replicas*

1

Upgrade system on first boot

Operating System Profile
swisscom-ubuntu-22.04

Leave this field blank to use default operating system profile.

CPU 4/16 Memory 16/32 GB Disk 20/500 GB



01 – Create the Kubernetes Cluster

Initial Nodes

- Deploy only one node
- Increase CPU to 4
- Increase Memory to 16 GB
- Increase Disk size to 20 GB

KubeVirt Settings

No Instance Types Available No Preferences Available Please select an instance type first.

View

CPUs* 4

Memory (MB)* 16000

Subnet*

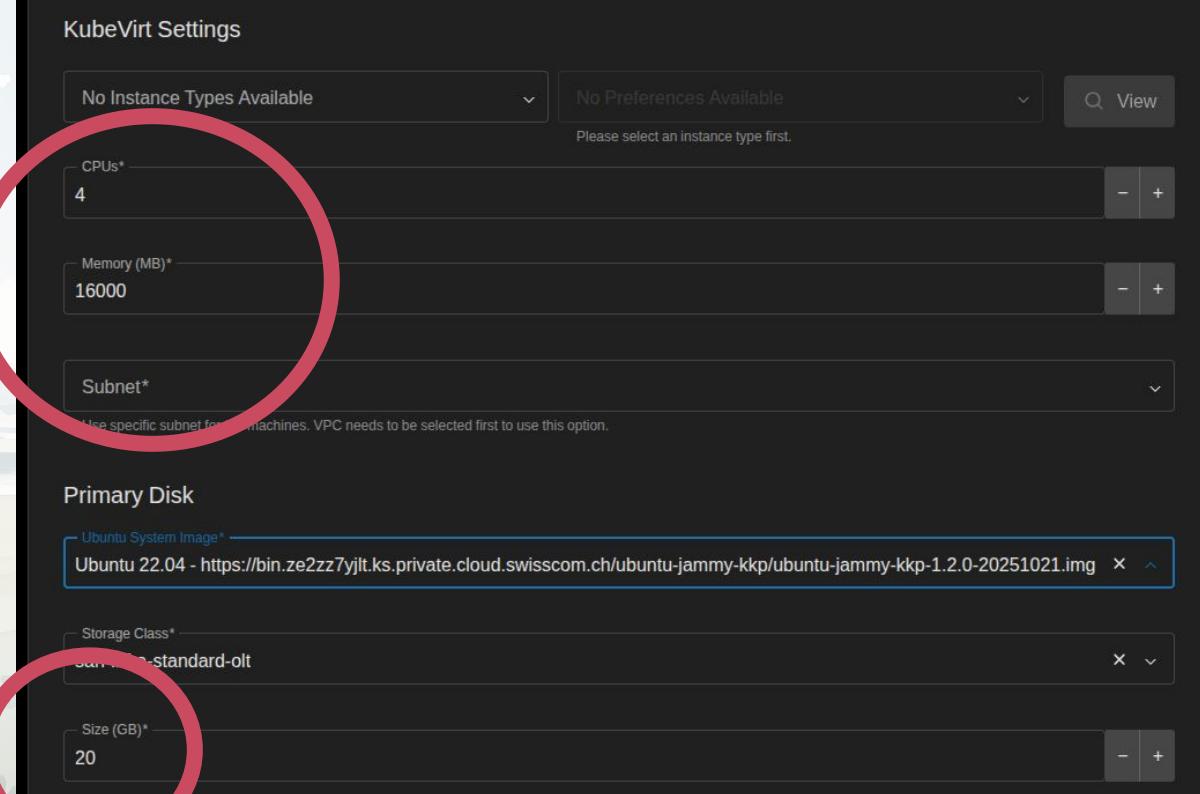
Use specific subnet for VMs. VPC needs to be selected first to use this option.

Primary Disk

Ubuntu System Image* Ubuntu 22.04 - https://bin.ze2zz7yjt.ks.private.cloud.swisscom.ch/ubuntu-jammy-kkp/ubuntu-jammy-kkp-1.2.0-20251021.img

Storage Class* swarm-standard-olt

Size (GB)* 20





01 – Create the Kubernetes Cluster

Initial Nodes

- Select any subnet
- Select a Primary disk

CPUs*
4

Memory (MB)*
16000

Subnet*
draco-491-zvs1k1-qh2oxu-olt-01654f (10.50.5.0/24)

Use specific subnet for the machines. VPC needs to be selected first to use this option.

Primary Disk

Ubuntu System Image*

Ubuntu 22.04 - <https://bin.ze2zz7yjt.ks.private.cloud.swisscom.ch/ubuntu-jammy-kkp/ubuntu-jammy-kkp-1.2.0-20251021.img>

Storage Class*
san-infra-standard-olt

Size (GB)*
20

ADVANCED SCHEDULING SETTINGS ▾

 Next

Cancel



01 – Create the Kubernetes Cluster

Applications

- Add Nginx

Create Cluster

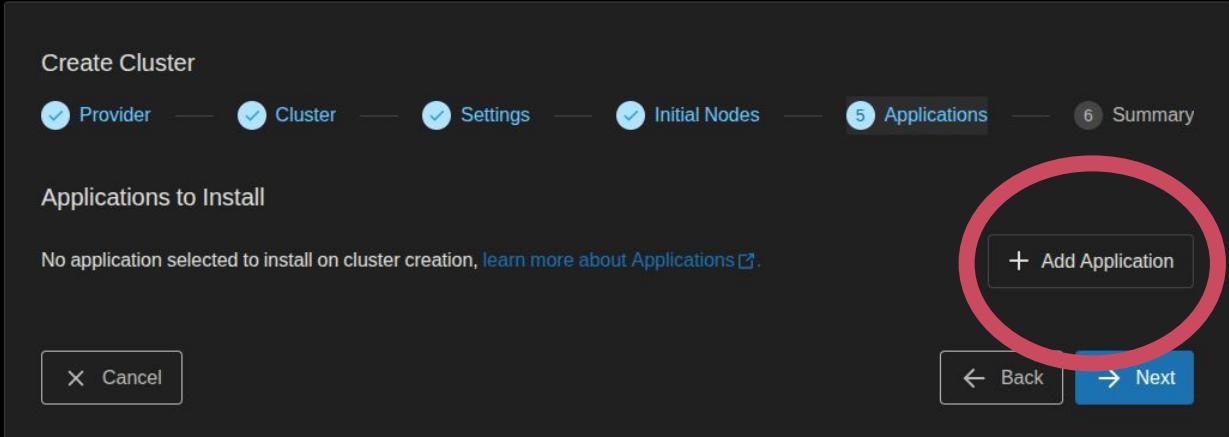
Provider Cluster Settings Initial Nodes Applications Summary

Applications to Install

No application selected to install on cluster creation, [learn more about Applications](#).

+ Add Application

Cancel Back Next





01 – Create the Kubernetes Cluster

Applications

- Add Nginx

The screenshot shows a 'Create Cluster' interface with a sidebar labeled 'Provider' and a main area titled 'Add Application'. The 'Select Application' step is active. The interface lists several applications:

- cloudnative-pg: CloudNativePG is a Kubernetes operator that covers the full lifecycle of a highly available PostgreSQL database cluster with a primary/standby architecture, using native streaming replication.
- csi-driver-nfs: CSI driver for Kubernetes to enable usage of NFSv3 or NFSv4 server shares for persistent volumes.
- Ingress-nginx: Ingress-NGINX is an Ingress controller for Kubernetes, using NGINX as a reverse proxy and load balancer. This item is highlighted with a red circle.
- longhorn: Longhorn is a distributed, cloud-native block storage system for Kubernetes, providing support for persistent volumes replicated across multiple replicas stored on multiple nodes.

Navigation buttons 'Back' and 'Next' are visible at the bottom right of the application list.

01 – Create the Kubernetes Cluster

Applications



- Add Nginx

Create Cluster

Provider

Applications

No application s

Cancel

Add Application

Select Application

Settings

Application Values

ingress-nginx

Ingress-NGINX is an Ingress controller for Kubernetes, using NGINX as a reverse proxy and load balancer.

Version*
4.13.3

Method
HELM

Source
HELM

Documentation - </> Source

Application Installation Namespace*
ingress-nginx

Namespace where application installation will be created.

Name*
ingress-nginx

Application Resources Namespace*
ingress-nginx

Namespace where application resources will be deployed.

Back

Next

ns 6 Summary

+ Add Application

Back Next

The screenshot shows a dark-themed UI for creating a Kubernetes cluster. On the left, a sidebar lists 'Create Cluster' and 'Provider'. Below that is a section for 'Applications' with the message 'No application s'. A 'Cancel' button is present. The main area is titled 'Add Application' and shows a step-by-step process: 'Select Application' (marked with a checkmark), 'Settings', and 'Application Values'. Under 'Select Application', there's a card for 'ingress-nginx' with its description: 'Ingress-NGINX is an Ingress controller for Kubernetes, using NGINX as a reverse proxy and load balancer.' It includes fields for 'Version*' (set to 4.13.3) and 'Method' and 'Source' both set to 'HELM'. Below this are fields for 'Application Installation Namespace*' (set to 'ingress-nginx') and 'Name*' (set to 'ingress-nginx'). Further down are fields for 'Application Resources Namespace*' (set to 'ingress-nginx') and 'Name*' (set to 'ingress-nginx'). At the bottom are 'Back' and 'Next' buttons, with 'Next' being highlighted by a large red circle.

01 – Create the Kubernetes Cluster

Applications



- Add Nginx

Create Cluster

Provider

Applications

No application s

X Cancel

Add Application

1 Select Application 2 Settings 3 Application Values

Optional: Use custom values to override the default configuration for this application.

values.yaml

```
1 global:
2   image:
3     registry: registry.ze2zz7yjlt.ks.private.cloud.swisscom.ch
4   controller:
5     metrics:
6       enabled: true
7     service:
8       type: LoadBalancer
9       loadBalancerClass: kubelb
10    annotations:
11      prometheus.io/scrape: "true"
12      prometheus.io/port: "10254"
13    ingressClassResource:
14      default: true
15    replicaCount: 2
16
```

Input should be valid YAML

Back Next

+ Add Application



01 – Create the Kubernetes Cluster

Applications

- Add Nginx
- Add Openmetrics
Particle
Accelerator Lab

Create Cluster

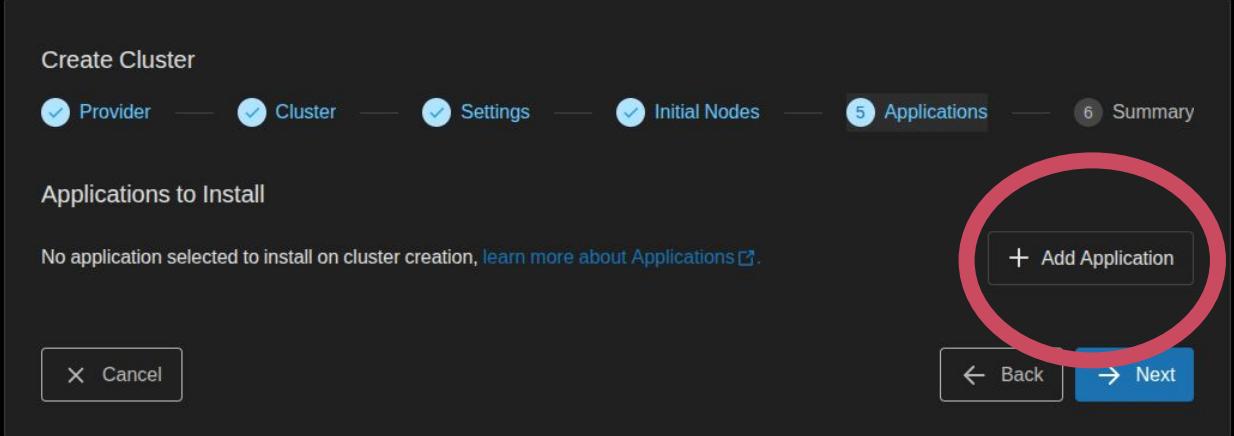
Provider Cluster Settings Initial Nodes Applications Summary

Applications to Install

No application selected to install on cluster creation, [learn more about Applications](#).

+ Add Application

Cancel Back Next



01 – Create the Kubernetes Cluster

Applications



- Add Nginx
- Add Openmetrics
Particle
Accelerator Lab

The screenshot shows a 'Create Cluster' interface with an 'Add Application' overlay. The overlay has six steps: 1. Select Application (NFS), 2. Settings, 3. Application Values, 4. Summary, 5. Options, and 6. Summary. Step 1 is completed. The 'Select Application' section lists three applications: 'ingress-nginx' (reverse proxy and load balancer), 'longhorn' (distributed cloud-native block storage system), and 'Openmetrics Particle Accelerator Lab'. The 'Openmetrics Particle Accelerator Lab' entry is circled in red.

Applications

No application selected

Cancel

Next

Provider

1 Select Application

NFS

ingress-nginx

Ingress-NGINX is an Ingress controller for Kubernetes, using NGINX as a reverse proxy and load balancer.

longhorn

Longhorn is a distributed, cloud-native block storage system for Kubernetes, providing support for persistent volumes replicated across multiple replicas stored on multiple nodes.

Openmetrics Particle Accelerator Lab

This project is a small experimental lab designed to demonstrate how easily OpenMetrics metrics can be integrated into a business application for Prometheus monitoring.

01 – Create the Kubernetes Cluster

Applications



- Add Nginx
- Add Openmetrics
Particle
Accelerator Lab

The screenshot shows a 'Create Cluster' interface with a 'Provider' selected. The main focus is the 'Add Application' step, which is the second in a three-step process. The application being added is 'Openmetrics Particle Accelerator Lab'. The version is set to 0.1.0. The 'Application Installation Namespace' is set to 'openmetrics-accelerator-lab'. The 'Name' field also contains 'openmetrics-accelerator-lab'. The 'Application Resources Namespace' is also set to 'openmetrics-accelerator-lab'. A large red arrow points to the 'Application Resources Namespace' input field.

Add Application

Create Cluster

Provider

Applications

No application selected

Cancel

Select Application

Settings

Application Values

Openmetrics Particle Accelerator Lab

This project is a small experimental lab designed to demonstrate how easily OpenMetrics metrics can be integrated into a business application for Prometheus monitoring.

Documentation - </> Source

Version*

0.1.0

Method Source

Application Installation Namespace*

openmetrics-accelerator-lab

Namespace where application installation will be created.

Name*

openmetrics-accelerator-lab

Application Resources Namespace*

openmetrics-accelerator-lab

Back

Next

camptocamp.com

01 – Create the Kubernetes Cluster Applications



- Add Nginx
- Add Openmetrics
Particle
Accelerator Lab

The screenshot shows a dark-themed UI for creating a Kubernetes application. On the left, a sidebar lists 'Create Cluster', 'Provider' (selected), 'Applications' (empty), and 'No application selected'. The main area is titled 'Add Application' and shows three steps: 'Select Application' (completed with a checkmark), 'Settings' (completed with a checkmark), and 'Application Values' (step 3). Below these steps is a note: 'Optional: Use custom values to override the default configuration for this application.' followed by a code editor containing a single line of YAML: '1 # empty'. At the bottom of the editor is the error message 'Input should be valid YAML'. At the very bottom of the modal are 'Back' and 'Next' buttons, and a prominent blue 'Add Application' button.



01 – Create the Kubernetes Cluster

Applications

- Add Nginx
- Add Openmetrics
Particle
Accelerator Lab

Create Cluster

Provider Cluster Settings Initial Nodes Applications Summary

Applications to Install

+ Add Application

Search

Application	Version	Actions
ingress-nginx	4.13.3	Edit Delete
Openmetrics Particle Accelerator Lab	0.1.0	Edit Delete

Cancel Next





01 – Create the Kubernetes Cluster

Summary

Create Cluster

Provider Cluster Settings Initial Nodes Applications Summary

① Provider
Provider: KubeVirt Datacenter: east (CH)

② Cluster

④ Initial Nodes
GENERAL
Name: Autogenerated name Replicas: 1 Operating System: Ubuntu
Operating System Profile: swisscom-ubuntu-22.04 Node Annotations:

③ Settings
Preset: swisscom-east

⑤ Applications
ingress-nginx
Deployment: 4.13.3 Helm: 4.13.3
↳ ingress-nginx



Cancel Back Save Cluster Template + Create Cluster

camptocamp.com

01 – Create the Kubernetes Cluster



Get Kubeconfig

- Download Kubeconfig
- Remember the location of the file for next steps

The screenshot shows the KubeSphere UI for managing clusters. At the top, there's a navigation bar with a cluster dropdown set to "particle-accelerator". Below it, the cluster details are shown: Control Plane version 1.33.5, Container runtime CNI version 1.18.2, Cluster name mh5pm5vkr8, and creation time "a few seconds ago". The UI includes sections for Region (CH (east)), Provider (KubeVirt), Preset (swisscom-east), Container Runtime (containerd), and SSH Keys (No assigned keys). A prominent red circle highlights the "Get Kubeconfig" button in the top right corner of the main cluster card. Below the cluster card, there's a section titled "ADDITIONAL CLUSTER INFORMATION" with tabs for Control Plane, Networking, and OPA. The Control Plane tab lists various components like API Server, etcd, Scheduler, Controller, Machine Controller, Operating System Manager, User Controller Manager, Kubernetes Dashboard (which is selected and highlighted in green), and Kubermatic KubeLB. The Networking tab shows Pod CIDR (172.26.0.0/16), Service CIDR (10.241.0.0/20), and Node CIDR Mask Size (24). The OPA tab shows Policy Control (disabled). At the bottom of the screenshot, the URL "camptocamp.com" is visible.

particle-accelerator

Control Plane 1.33.5

CNI 1.18.2

Cluster mh5pm5vkr8 Created a few seconds ago

Region CH (east) Provider KubeVirt Preset swisscom-east Container Runtime containerd SSH Keys No assigned keys

ADDITIONAL CLUSTER INFORMATION

Control Plane	Networking	OPA
API Server	Proxy Mode ebpf	Policy Control
etcd	Expose Strategy Tunneling	
Scheduler	Pods CIDR 172.26.0.0/16	
Controller	Services CIDR 10.241.0.0/20	
Machine Controller	Node CIDR Mask Size 24	
Operating System Manager		
User Controller Manager		
Kubernetes Dashboard	✓ Node Local DNS Cache	
Kubermatic KubeLB	✓ Konnectivity	

camptocamp.com



01 – Create the Kubernetes Cluster

Summary

Instructions on
lab.campto.com

- Sign-in using OIDC
- Create Resource → Cluster
- 1. Provider: Choose default provider and datacenter
- 2. Cluster: Set cluster name
- 3. Settings: Choose default settings
- 4. Initial Nodes: One node with 4 CPUs, 16 GB of RAM and 20 GB of disk space
- 5. Applications: Deploy Nginx and the Lab



02 – Install tools

kubectl

- Cluster nodes are not reachable



Kubernetes Nodes



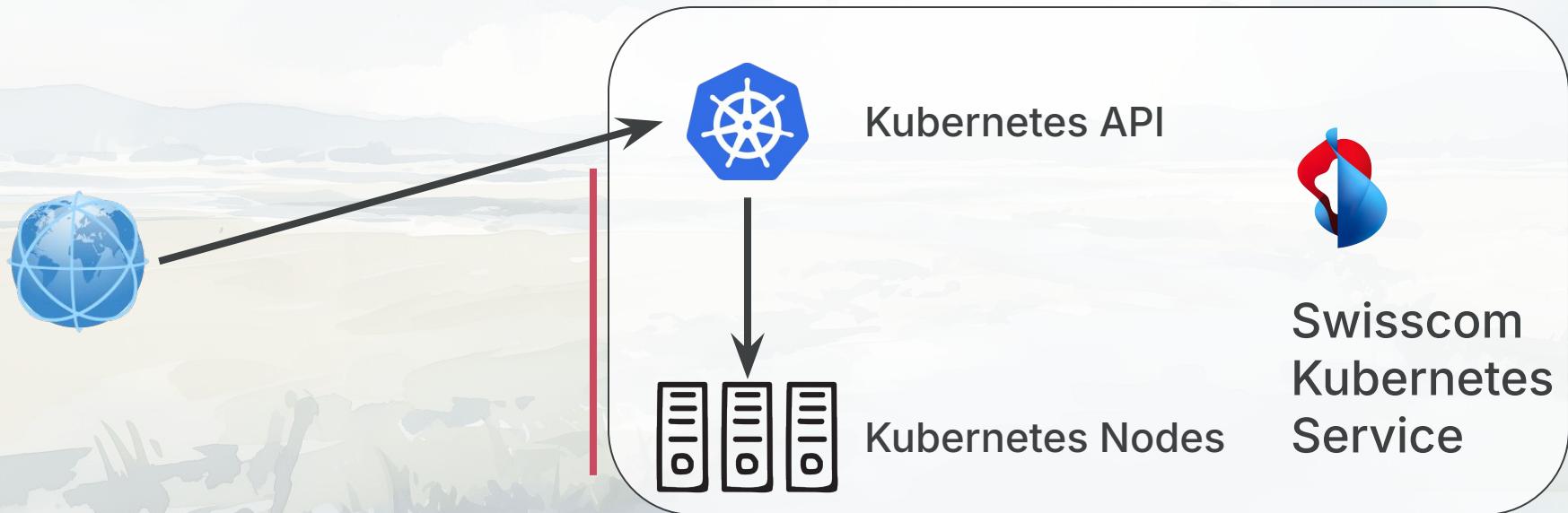
Swisscom
Kubernetes
Service



02 – Install tools

kubectl

- Cluster nodes are not reachable
- Port forward to an ingress controller pod





02 – Install tools

kubectl

- Follow official instructions
- Phase 1 – "Install Kubectl" on lab.campto.camp
- API version v1.33.5
- Kubernetes version skew policy -1/+1:
 - 1.32
 - 1.33
 - 1.34 (latest)



02 – Install tools

kubectl

- Linux:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

- macOS:

```
brew install kubectl
```

- Windows:

```
curl.exe -LO "https://dl.k8s.io/release/v1.34.2/bin/windows/amd64/kubectl.exe"
```

```
$ kubectl version --client  
Client Version: v1.34.2
```



02 – Install tools

Configure API Access

- Locate `kubeconfig-xxxxxxxxxxxx`
- Setup path to the `kubeconfig` file

The screenshot shows a cluster configuration page. At the top, there is a dropdown menu set to 'particle-accelerator'. To the right of the dropdown are two buttons: 'Get Kubeconfig' (highlighted with a red circle) and 'Open Dashboard'. Below this, there are several cluster details: Control Plane version 1.33.5, CNI version 1.18.2, Region CH (east), Provider KubeVirt, Preset swisscom-east, Container Runtime containerd, and SSH Keys (no assigned keys). At the bottom, there is a link 'ADDITIONAL CLUSTER INFORMATION ^'.

```
# Linux/Mac
```

```
export KUBECONFIG=~/Downloads/kubeconfig-xxxxxxxxxxxx
```

```
# Windows PowerShell
```

```
$env:KUBECONFIG="C:\Users\YourUser\Downloads\kubeconfig-xxxxxx"
```



02 – Install tools

Validate Access

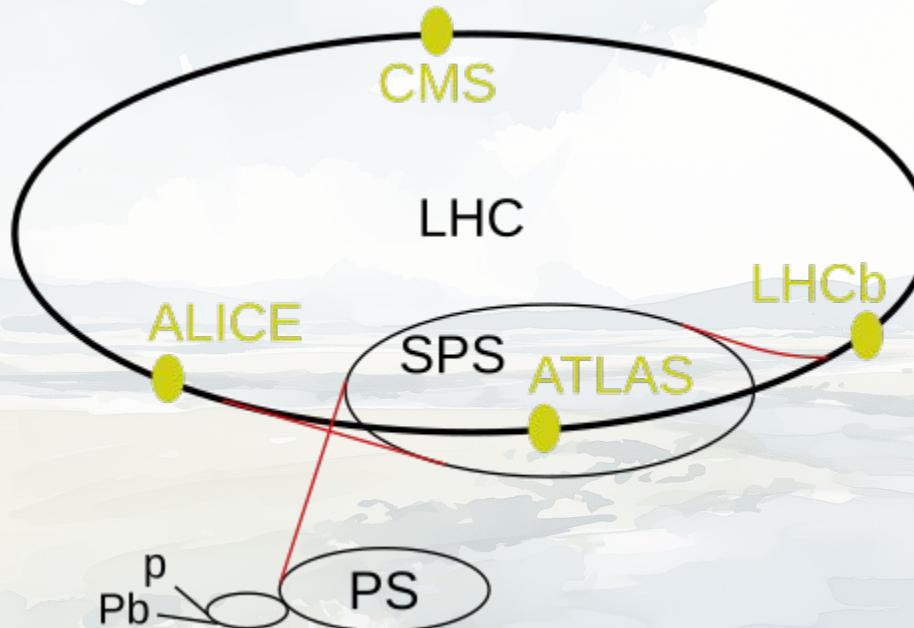
```
$ kubectl get pod -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
ingress-nginx	ingress-nginx-controller-549886-cpg2d	1/1	Running	0	6h42m
ingress-nginx	ingress-nginx-controller-549886-plbz5	1/1	Running	0	6h42m
kube-system	cilium-m7wf6	1/1	Running	0	10h
kube-system	cilium-operator-74668d995-hrpjh	1/1	Running	0	10h
kube-system	coredns-5d6489bb45-n4rrx	1/1	Running	0	10h
kube-system	coredns-5d6489bb45-v6mt2	1/1	Running	0	10h
kube-system	envoy-agent-jndgw	2/2	Running	0	10h
kube-system	hubble-relay-795945d857-xgqvw	1/1	Running	0	10h
kube-system	hubble-ui-556747b9c9-q6q29	2/2	Running	0	10h
kube-system	konnectivity-agent-7f7d9474dd-4pxjv	1/1	Running	0	10h
kube-system	konnectivity-agent-7f7d9474dd-gkp5s	1/1	Running	0	10h
kube-system	metrics-server-7bd6766f9d-5grc8	1/1	Running	0	10h
...					



03 – Python based Particle Accelerator

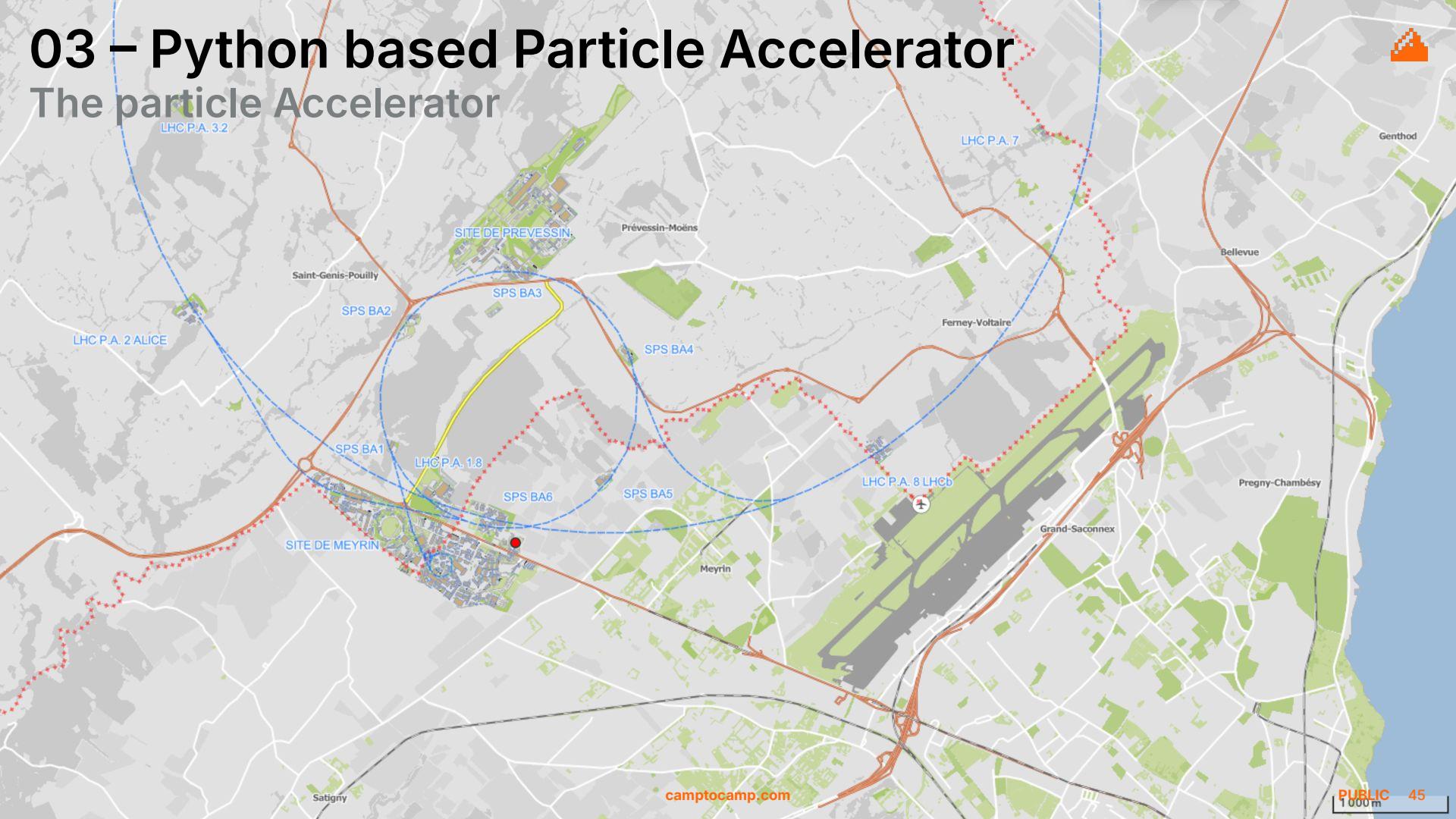
The particle Accelerator





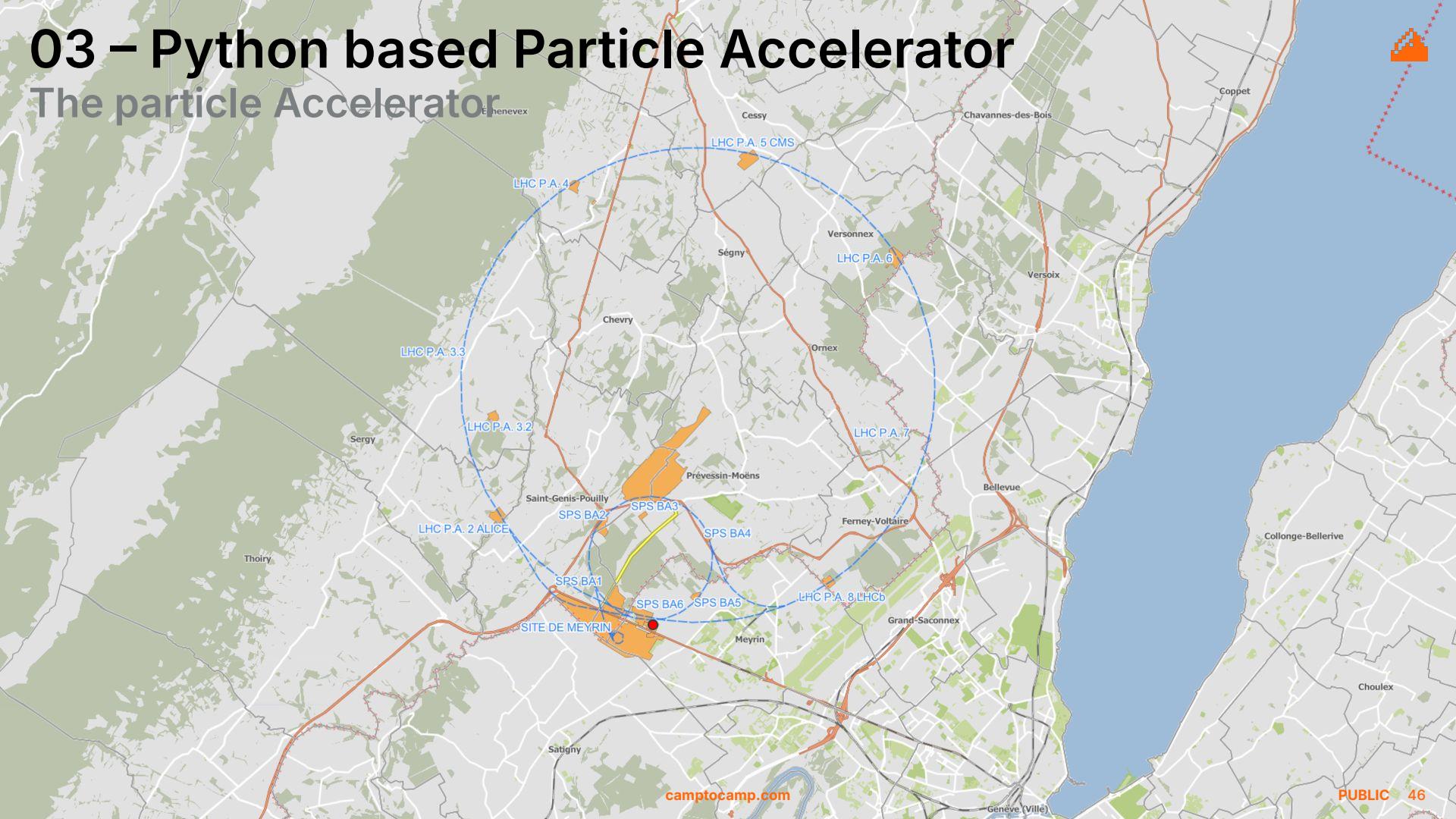
03 – Python based Particle Accelerator

The particle Accelerator



03 – Python based Particle Accelerator

The particle Accelerator



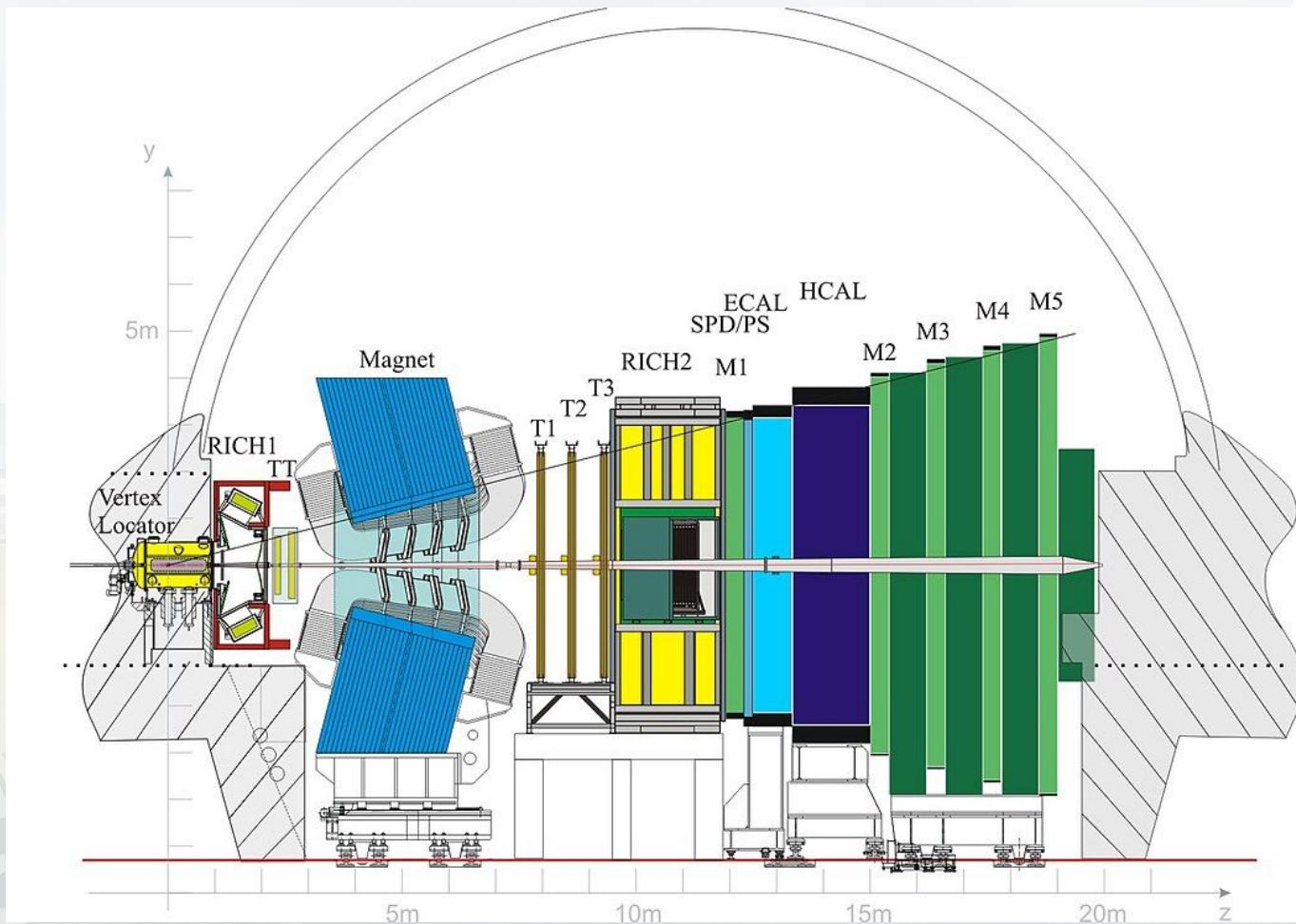


03 – Python based Particle Accelerator

The particle Accelerator



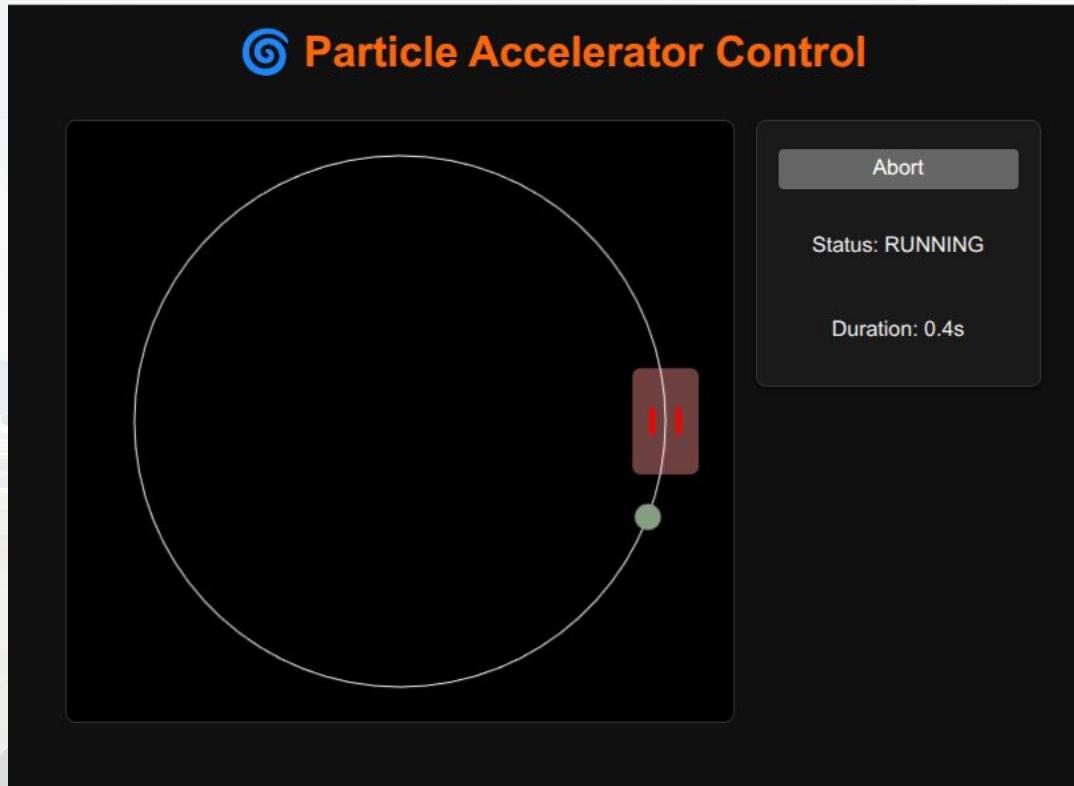
03 – Python based Particle Accelerator





03 – Python based Particle Accelerator

The very simple python implementation



03 - Python based Particle Accelerator



- Goal:
 - Collide protons to find new sub particles
 - Accelerate protons faster
 - 0.99999999% speed of light (nine 9s)
- The "Kick":
 - Particles orbit the ring 11,000 times/second.
 - RF cavity provides an electric "kick" to increase energy.
- Accelerator challenge:
 - Too Weak: Beam won't reach collision energy (~6.8 TeV).
 - Too Strong: Superconducting magnets overheat.

04 – Deploy the lab



- The Stack:
 - Cluster: Ephemeral Kubernetes Environment.
 - App: Python (Flask) simulating the accelerator physics.
 - Observability: OpenMetrics (client library) → Prometheus → Grafana.
- The Workflow:
 - Deploy: Launch the lab via the dashboard.
 - Port Forward: Establish a secure tunnel to the cluster.

04 – Deploy the lab

Establish Uplink (Port Forward)



- The Kubernetes cluster runs in an isolated network
- Single port forward to the ingress controller

```
$ kubectl -n ingress-nginx port-forward deploy/ingress-nginx-ingress-nginx-controller 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

This exposes all workshop services to your local machine on port 8080.

! If this command stops (Ctrl-C, terminal closed, laptop sleeps), you lose access to the lab.

04 – Deploy the lab

The Feedback Loop



- Launch simulator.
- Watch Grafana Dashboards.
- Change Code in Browser (VSCode)
- Save (Auto-reload).
- React to the data.

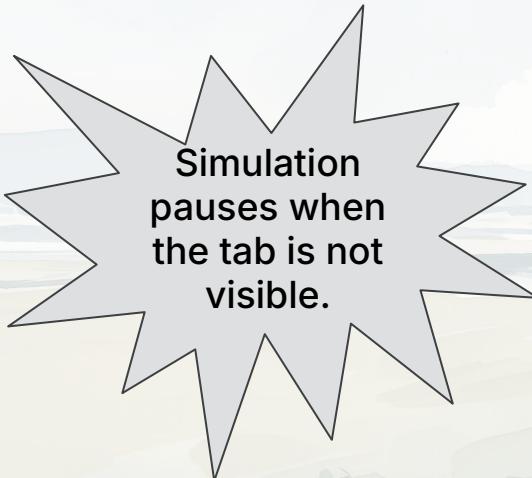


04 – Deploy the lab

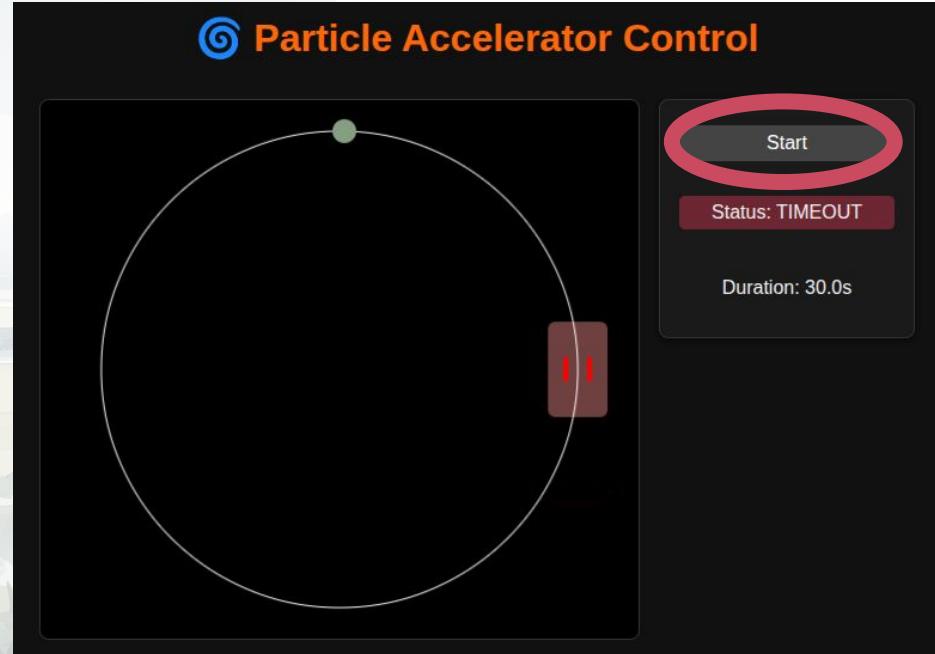
The Feedback Loop



- Launch simulator



<http://lab.127.0.0.1.nip.io:8080/>

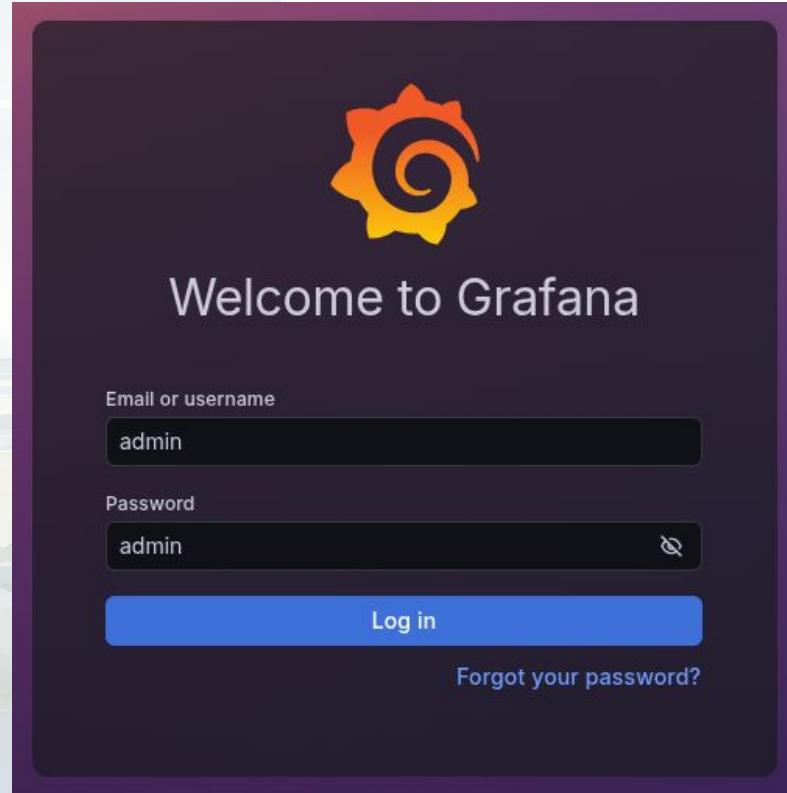


04 – Deploy the lab

The Feedback Loop

- Launch simulator
- Watch Grafana

<http://grafana.127.0.0.1.nip.io:8080/>



04 – Deploy the lab

The Feedback Loop



- Launch simulator
- Watch Grafana

<http://grafana.127.0.0.1.nip.io:8080/>

Update your password

Continuing to use the default password exposes you to security risks.

New password

Confirm new password

Submit

Skip

04 – Deploy the lab

The Feedback Loop



<http://grafana.127.0.0.1.nip.io:8080/>

- Launch simulator
- Watch Grafana

The screenshot shows the Grafana interface with a dark theme. On the left, a sidebar menu includes Home, Bookmarks, Starred, **Dashboards**, Explore, Drilldown, Alerting, Connections, and Administration. The 'Dashboards' item is highlighted with a red oval. The main content area is titled 'Dashboards' and contains the sub-instruction 'Create and manage dashboards to visualize your data'. It features a search bar ('Search for dashboards and folders'), a filter dropdown ('Filter by tag'), and a checkbox for 'Starred'. A list of dashboards is shown, with the first one, 'Openmetrics Accelerator Lab', also circled in red.

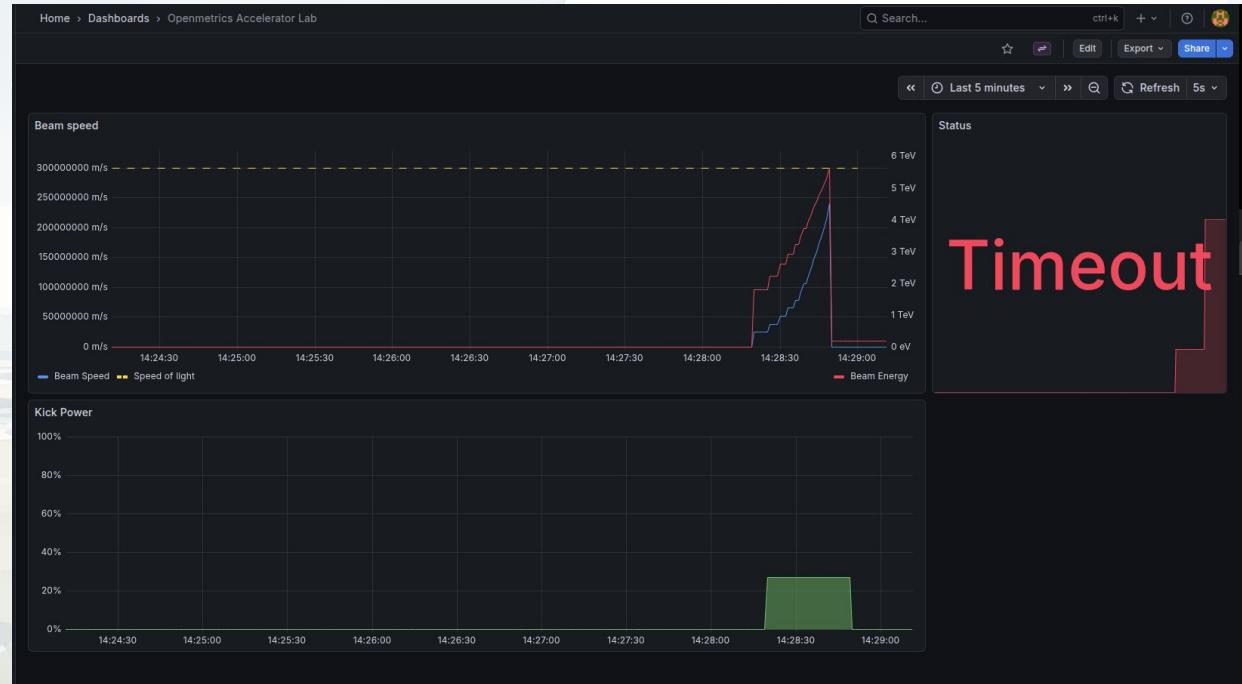
04 – Deploy the lab

The Feedback Loop



- Launch simulator
- Watch Grafana

<http://grafana.127.0.0.1.nip.io:8080/>

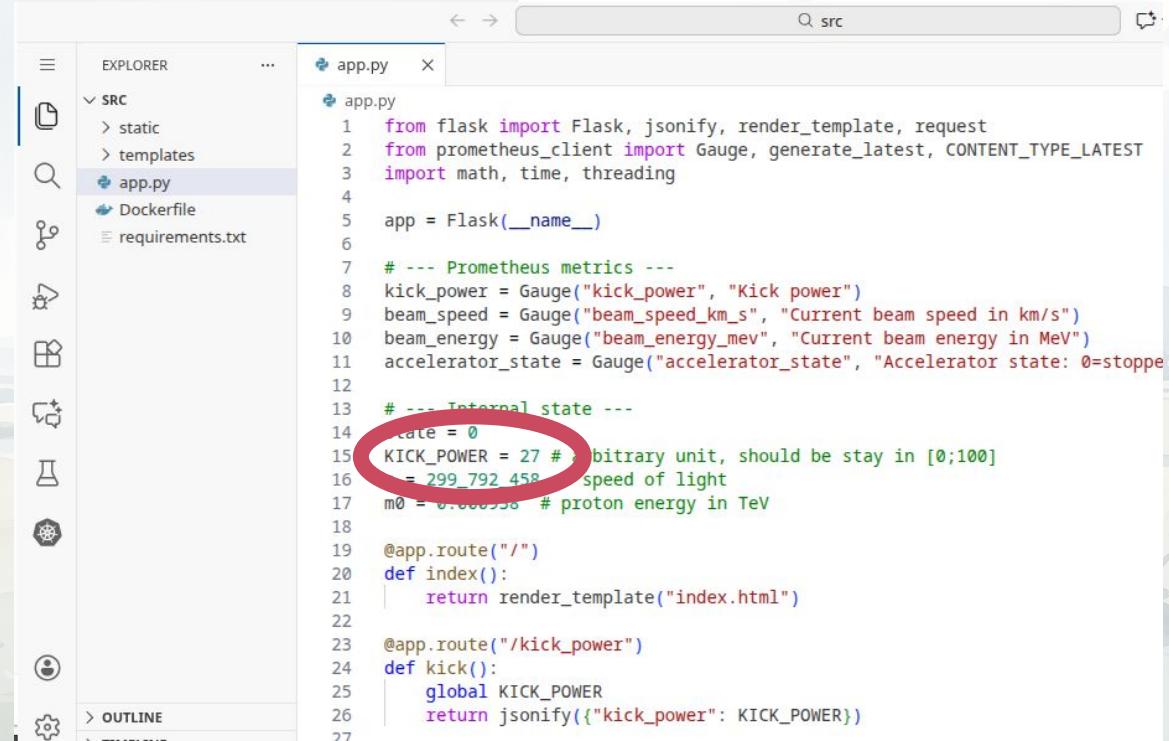


04 – Deploy the lab

The Feedback Loop

- Launch simulator
- Watch Grafana
- Change Code

<http://vscode.127.0.0.1.nip.io:8080/>



The screenshot shows the VS Code interface with the Explorer sidebar open, displaying files in the SRC folder: static, templates, app.py (selected), Dockerfile, and requirements.txt. The main editor tab shows the content of app.py. A red circle highlights the line `KICK_POWER = 27`. The code is as follows:

```
from flask import Flask, jsonify, render_template, request
from prometheus_client import Gauge, generate_latest, CONTENT_TYPE_LATEST
import math, time, threading
app = Flask(__name__)

# --- Prometheus metrics ---
kick_power = Gauge("kick_power", "Kick power")
beam_speed = Gauge("beam_speed_km_s", "Current beam speed in km/s")
beam_energy = Gauge("beam_energy_mev", "Current beam energy in MeV")
accelerator_state = Gauge("accelerator_state", "Accelerator state: 0=stoppe
...
# --- Internal state ---
rate = 0
KICK_POWER = 27 # arbitrary unit, should be stay in [0;100]
c = 299_792_458 # speed of light
m0 = 0.000538 # proton energy in TeV

@app.route("/")
def index():
    return render_template("index.html")

@app.route("/kick_power")
def kick():
    global KICK_POWER
    return jsonify({"kick_power": KICK_POWER})
```

04 – Deploy the lab

Command Center



The particle view

<http://lab.127.0.0.1.nip.io:8080/>



The Data View (Grafana)

<http://grafana.127.0.0.1.nip.io:8080/>



The Code View (VSCode)

<http://vscode.127.0.0.1.nip.io:8080/>



04 – Deploy the lab

Restart Lab

- Open Dashboard
- Select Deployments (left panel)
- Select 'openmetrics...' namespace (top drop list)
- Select 'openmetrics-accelerator-lab-x-xx-lab'
- Click on restart ↺



04 – Deploy the lab

Application observability

- Open Dashboard
- Select Pod (left panel)
- Select 'openmetrics...' namespace (top drop list)
- Inject memory and cpu usage

The screenshot shows the KubeSphere UI for a cluster named 'particle-accelerator'. At the top, there are dropdown menus for Control Plane (1.33.5), CNI Plugin (cilium 1.18.2), Cluster ID (mh5pm5vkr8), and a timestamp (1 day ago). Below these are fields for Region (CH (east)), Provider (KubeVirt), Preset (swisscom-east), Container Runtime (containerd), and SSH Keys (No assigned keys). A 'Get Kubeconfig' button is also present. A large red circle highlights the 'Open Dashboard' button. To the right, there's a 'Web Terminal' button and a three-dot menu. The main area displays 'ADDITIONAL CLUSTER INFORMATION' with sections for Control Plane (API Server, etcd, Scheduler, Controller, Machine Controller, Operating System Manager, User Controller Manager) and Networking (Proxy Mode: ebpf, Expose Strategy: Tunneling, Pods CIDR: 172.26.0.0/16, Services CIDR: 10.241.0.0/20, Node CIDR Mask Size: 24). A green status indicator for 'Kubernetes Dashboard' is shown, along with two checked items: 'Node Local DNS Cache' and 'Konnectivity'.

04 – Deploy the lab



Extra step – Implements a new metrics

- kick_count
- Type: [counter](#)
- Call inc() in the kick() function
- Check metrics on [Prometheus](#)

```
1 from prometheus_client import Counter  
2  
3 kick_count = Counter('kick_count', 'Kick Count')  
4  
5 @app.route('/kick_power')  
6 def kick():  
7     ...  
8     kick_count.inc()  
9     ...
```



05 – Observability tools using eBPF

eBPF

- Small sandboxed program running in the Linux Kernel
- Custom programs injected at runtime in kernel
- No need to recompile or load modules
- Kernel feature started on 2011 with BPF: Berkeley Packet Filter
- 2014 Kernel 3.15 : Introduce eBPF support
- Many evolutions with Kernel 4, 5 and 6



05 – Observability tools using eBPF

eBPF

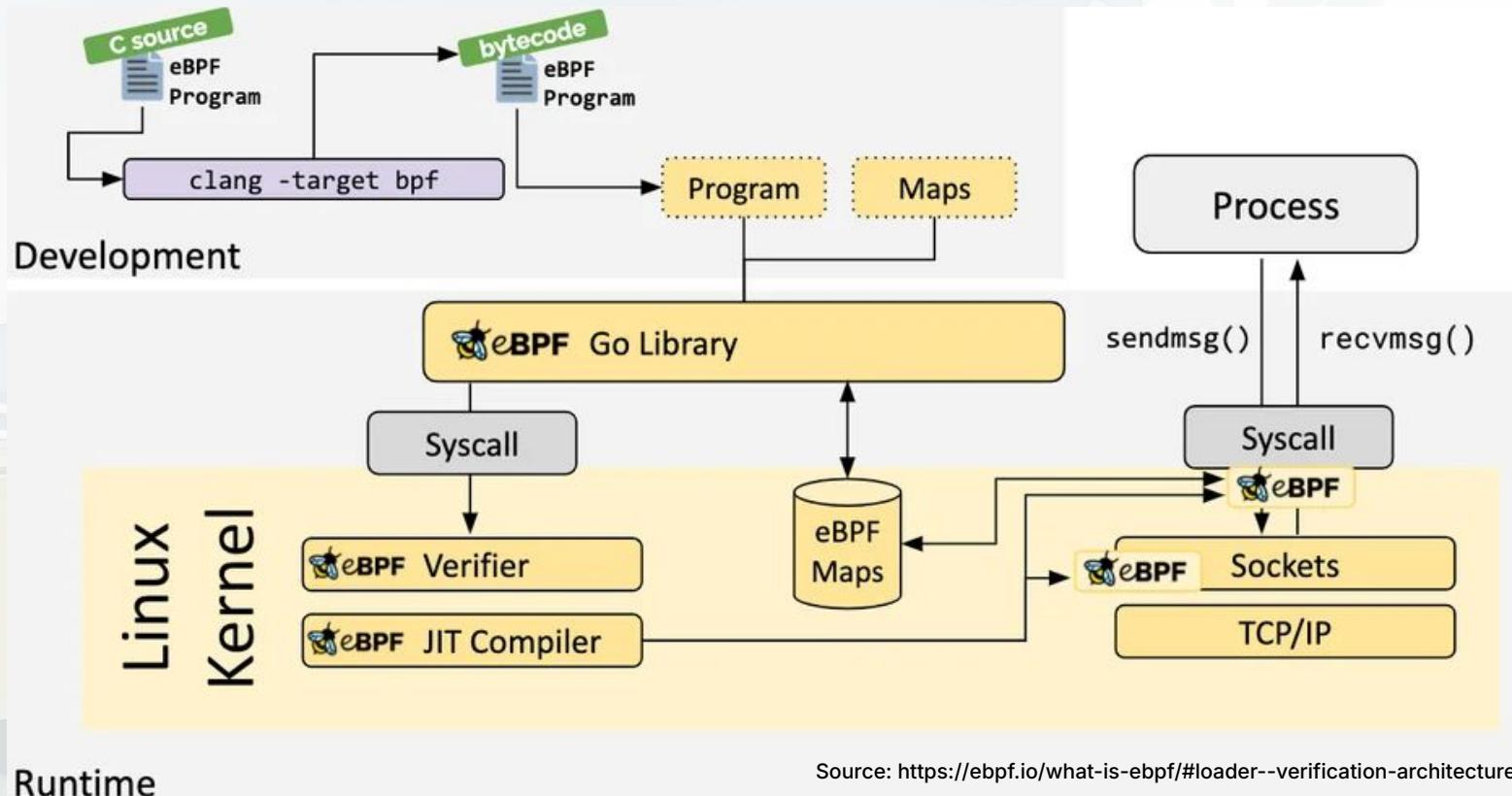
- Focus on stability and security
- User writes eBPF program with C language
- The verifier checks that:
 - user is allowed to inject eBPF programs
 - the program always run to completion
- Maps are used to exchange data between Kernel and user-space



05 – Observability tools using eBPF



eBPF



Runtime

05 – Observability tools using EBPF



- BPTrace : <https://github.com/bpftrace/bpftrace/tree/master/tools>
- Inspektor Gadget <https://inspektor-gadget.io/docs/latest/gadgets/>

```
$ kubectl debug --profile=sysadmin $(kubectl get node -o name) -ti \  
  --image=ghcr.io/inspektor-gadget/ig:latest -- \  
  ig run trace_exec:latest \  
  --filter k8s.namespace==openmetrics-accelerator-lab
```



Federico Sismondi

federico.sismondi@camptocamp.com

Julien Acroute

julien.acroute@camptocamp.com

Thanks for your attention.