Self-healing Clusters

Game of Nodes and Scaling the Throne

DESCHEDULER

Who am I?

Tyler Gillson

@tylergillson

https://linkedin.com/in/tyler-gillson

- Hands on with Kubernetes since 2019
- Enjoy building distributed systems and developing POCs
- Avid climber
- Principal Software Engineer @ Spectro Cloud

spectro cloud

Agenda

1. The Challenges at Hand
2. The Heroes of the Story
3. Demo

Challenges at Hand

Stability is key

- Downtime is not an option for mission-critical workloads on Kubernetes
  - AI/ML
  - Medical imaging
  - Video streaming
- As clusters grow, stability becomes a challenge
  - More nodes and pods can lead to management complexity and growth pains
  - How to prevent service outages or degradations?
  - Pods are probably all receiving BestEffort QoS

What makes a cluster unstable?

- **Pod Eviction**: Low node resources (pressure) leads to disruptions (kubelet)
- **Pod Preemption**: Excess pods lead to disruptions (kube-scheduler)
- **Resource Quotas**: Improper configurations can cause pod failures
- **Network Policies**: Incorrect settings disrupt pod communication
- **Stateful Applications**: Mismanagement can result in data loss
- **Logging and Monitoring**: Inadequate setups delay issue detection

Building blocks for stability

- Automated low-level monitoring
  - Node Problem Detector for real-time health checks
- Topology management
  - Cluster Autoscaler (CA) to adapt cluster size
  - Descheduler for balancing workloads
  - KEDA for scaling workloads to zero
  - Vertical Pod Autoscaler for optimizing resource allocation per pod
    - `InPlacePodVerticalScaling` (v1.27+, alpha, #4016)
  - Cluster Proportional Autoscaler (beta)

Building blocks for stability

- Policy enforcement
  - Pod Security Admission + Pod Security Standards are insufficient (v1.25+)
  - PaC: Kyverno, OPA/Gatekeeper, jsPolicy
- Logging and observability
  - Cluster-level logging (Fluentd, etc.)
  - Prometheus + Grafana
- Chaos engineering
  - ChaosMesh for resiliency testing

Heroes of the Story

The Three-Eyed Raven: Node Problem Detector

- Runs as a DaemonSet
- NPD leverages **Events** and **NodeConditions** to report problems to the apiserver
  - Events are native Kubernetes objects
  - NodeConditions are contained within a Node's status
- **Events** describe temporary or less severe issues
- **NodeConditions** register more persistent or severe health issues for a node
- Exporters report problems and/or metrics to various backends (kube-apiserver, Prometheus, Stackdriver)

The Three-Eyed Raven: Node Problem Detector

- Multiple problem daemons (AKA, sub-daemons) run within the NPD binary to monitor various issue types:
  - SystemLogMonitor: monitor kernel, container runtime logs (e.g., KernelDeadlock)
  - HealthChecker: monitor kubelet, container runtime health (e.g., KubeletUnhealthy, ContainerRuntimeUnhealthy)
  - CustomPluginMonitor: execute custom scripts (e.g., NTPProblem)
  - SystemStatsMonitor: system metrics collection (metrics only, used with the Prometheus exporter)

The Hand of the King: Descheduler

- The Kubernetes scheduler does not automatically evict Pods for rebalancing purposes
- Descheduler's policy-based eviction can rebalance a cluster
  - Prevents bottlenecks
  - Enhances cluster efficiency & saves $$$
- Can be run as a **Job**, **CronJob**, or **Deployment**
- Installed using Helm or Kustomize

The Hand of the King: Descheduler

- Multiple top-level policies are available (plugins)
  - **LowNodeUtilization**: Evict pods from overutilized nodes
  - **HighNodeUtilization**: Evict pods from underutilized nodes
  - **RemoveDuplicates**: Evict duplicate pods running on the same node
  - **RemovePodsViolatingInterPodAntiAffinity**
  - **RemovePodsViolatingNodeAffinity**
  - **RemovePodsViolatingNodeTaints**
    - Combine with NPD and CA to automatically remove Nodes experiencing issues
    - Only works for PIDPressure, MemoryPressure, DiskPressure, Ready, and some cloud provider specific conditions (will be resolved in #565)

The Master of Whisperers: Cluster Autoscaler (CA)

- Operational Details
  - Runs on the Kubernetes Control Plane
  - Typically via a Kubernetes Deployment
  - Consider your NodeResourcesFit scheduler plugin strategy (MostAllocated)

- Cluster Management
  - Dynamically adjusts cluster size, adding or removing nodes from node groups
  - Node and Pod exclusion via annotations

```
"cluster-autoscaler.kubernetes.io/safe-to-evict[-local-volumes]": "[true|false]"
"cluster-autoscaler.kubernetes.io/enable-ds-eviction": "true"
"cluster-autoscaler.kubernetes.io/scale-down-disabled": "true"
```

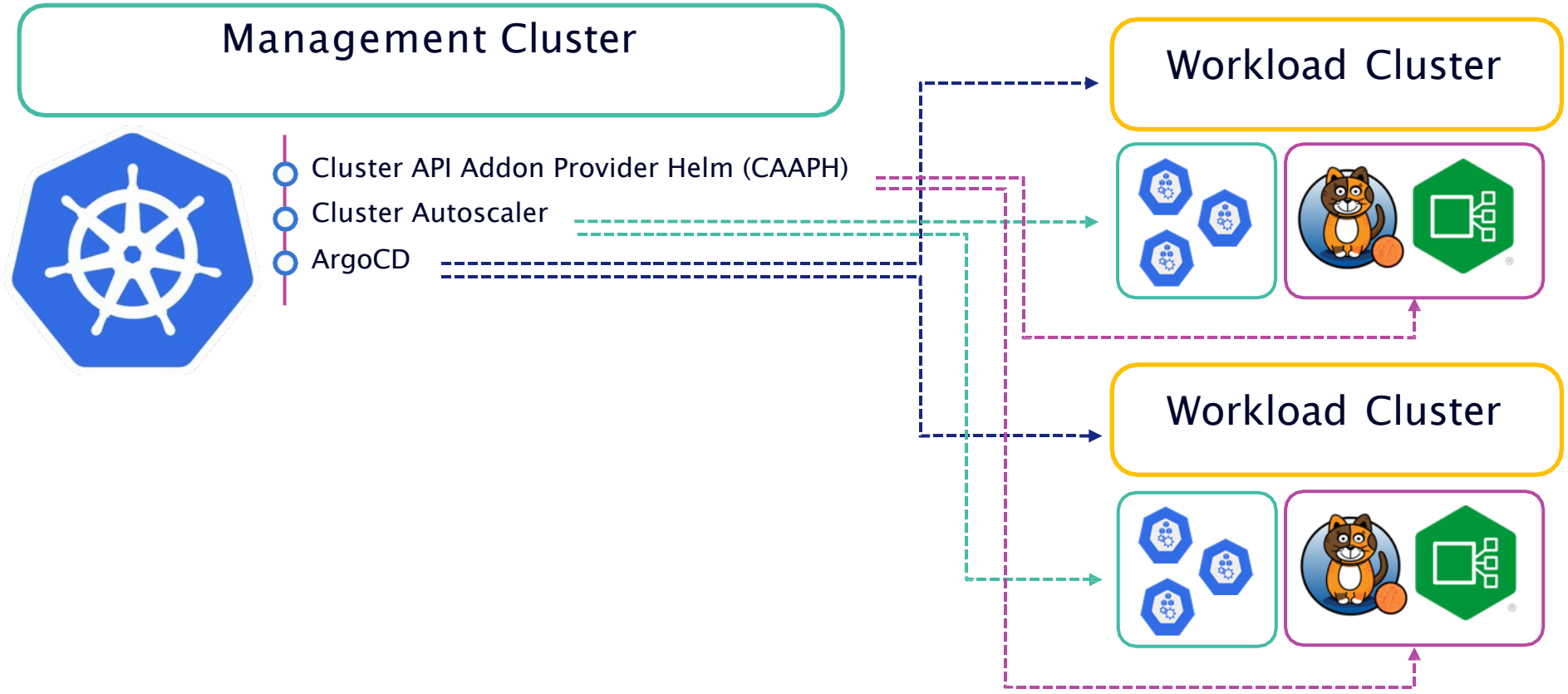
  - Pod exclusion via Priority Classes + priority cutoff
    - Pods with priority < -10 don't trigger scale-ups or prevent scale-downs

The Master of Whisperers: Cluster Autoscaler (CA)

- Scaling Intelligence
  - Scales up node groups based on pending/unschedulable pods
    - Expanders provide strategies for node group selection:
      `random, most-pods, least-waste, price, priority`
  - Scales down nodes having low (enough) resource requests, movable pods, and no blocking annotations for >10min (default)
    - SUM(CPU + Memory requests) below configurable threshold

- Interoperability and Extensibility
  - Compatible with 25+ Cloud Providers
  - Supports Cluster API (CAPI)

Example with CAPI



Example with CAPI

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cluster-autoscaler
  namespace: kube-system
spec:
  selector:
    matchLabels:
    app: cluster-autoscaler
```

Example with CAPI

```
template:
  spec:
    containers:
    - name: cluster-autoscaler
      args:
      - --kubeconfig=/mnt/value
      - --clusterapi-cloud-config-authoritative
      - --cloud-provider=clusterapi
      - --node-group-auto-discovery=clusterapi:clusterName=capi-dev
      volumeMounts:
      - name: kubeconfig-vol
        mountPath: /mnt

    volumes:
    - name: kubeconfig-vol
      secret:
         secretName: capi-dev-kubeconfig
```

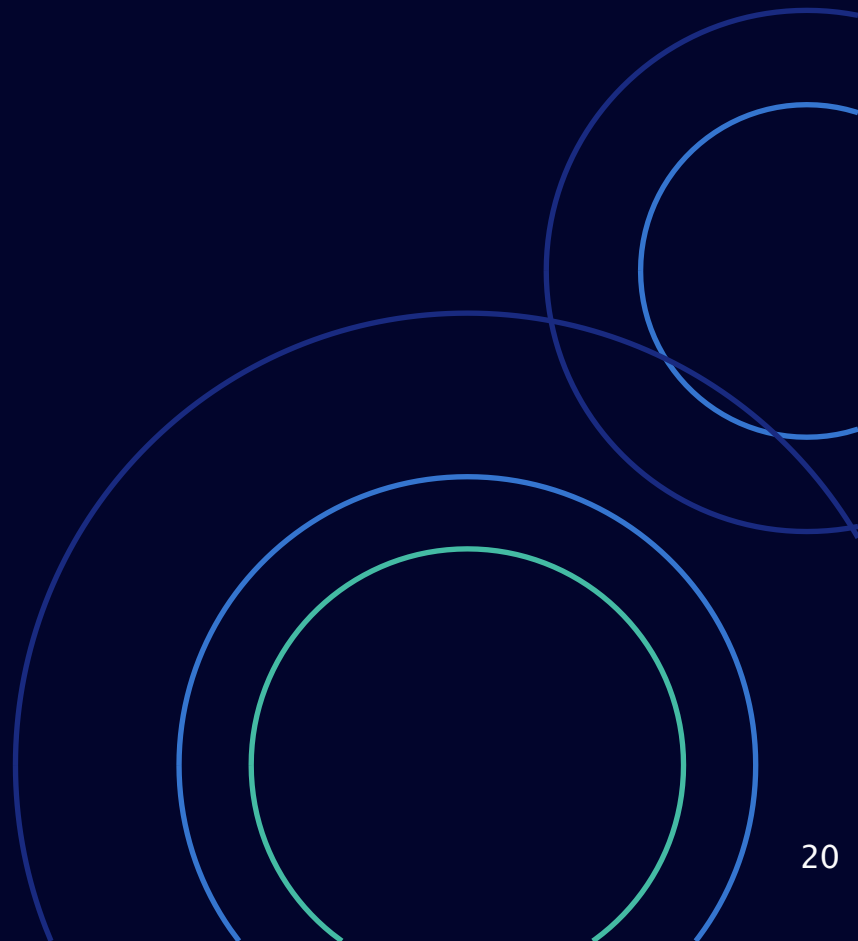
Example with CAPI

- Annotate the CAPI resource (**MachineSet/MachineDeployment/MachinePool**) with the following key/value pairs:

```
cluster.x-k8s.io/cluster-api-autoscaler-node-group-max-size: "10"
cluster.x-k8s.io/cluster-api-autoscaler-node-group-min-size: "1"
```

- Scale from zero
  - Native support in some, but not all, CAPI providers
  - You can still use *any* provider via capacity annotations

Let’s make them work together!



Workflow

- Deploy enough Pods to create resource pressure
- Watch as **CA** provisions a new node, **Descheduler** rebalances pods
- Update **Descheduler** config & delete Pods
- Watch as Pods are bin-packed, **CA** deprovisions the new node
- Test **NPD** by writing to `/dev/kmsg`
- Verify node conditions are updated, events created

-------- *Time Permitting* --------

- Manually stress one of the nodes
- Wait for the node controller to add a NoSchedule taint
- Watch **Descheduler** evict the pods and **CA** trigger a new node creation

spectro cloud

Key Takeaways

- Stability is the cornerstone of a resilient Kubernetes cluster
- Node Problem Detector, Descheduler, and Cluster Autoscaler play unique but complementary roles
- Be proactive, not reactive, by employing intelligent monitoring and rebalancing strategies
- Combine PDBs, scoped ResourceQuotas, and LimitRanges for a robust cluster
- Leverage the power of the Kubernetes API for declarative cluster lifecycle management

spectro cloud

palette

Manage new and existing single-cluster or multi-cluster, multi-distro Kubernetes environments from any location

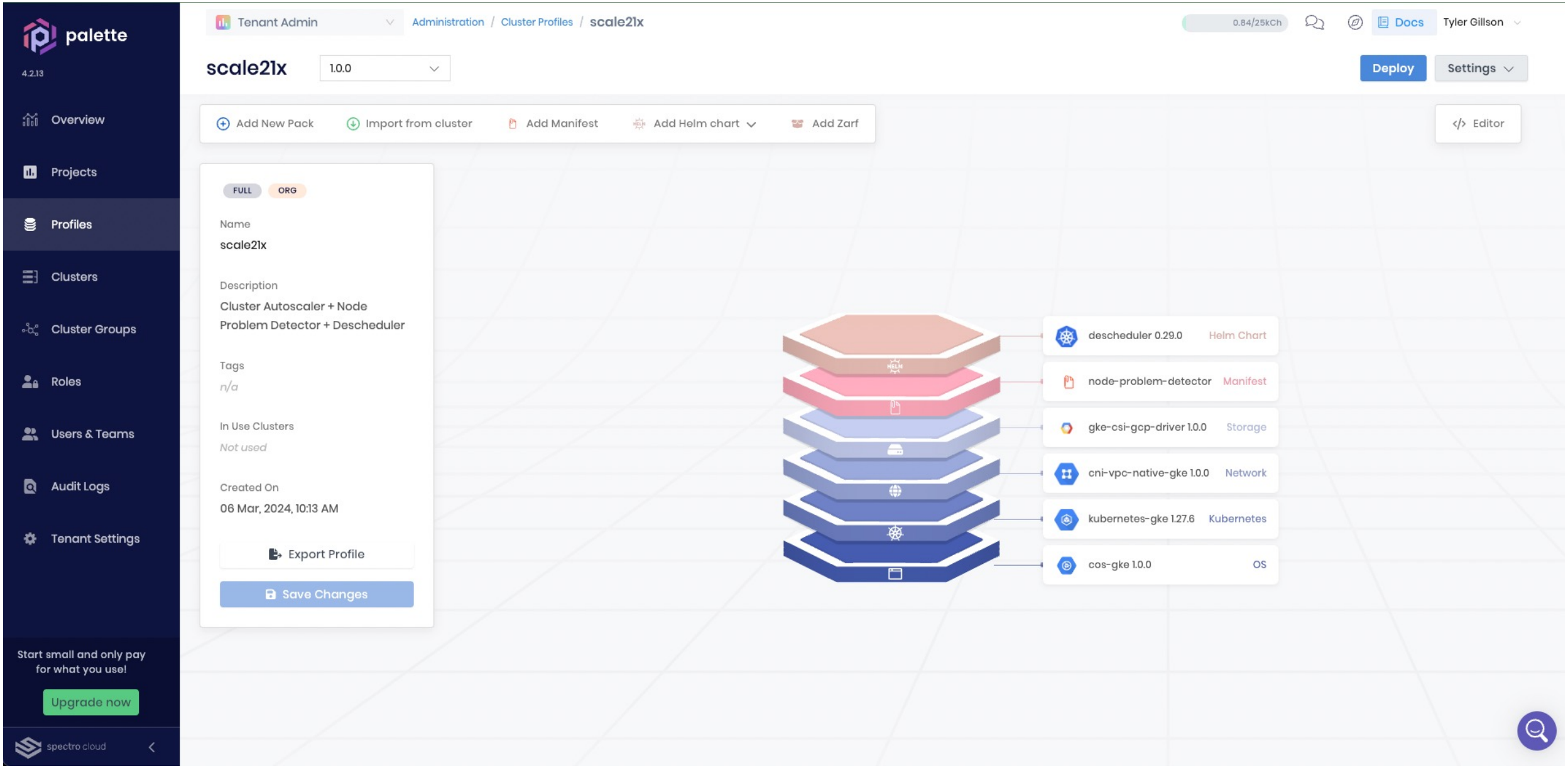
Email

I agree with the terms and conditions and give consent for my data to be used according to the privacy policy

Sign up

Already have an account? → **Sign in**

It's time for a computing platform
without boundaries



spectro cloud