15 MAY 2022

Cloud Native Rejekts:
Dealing with remote worker nodes

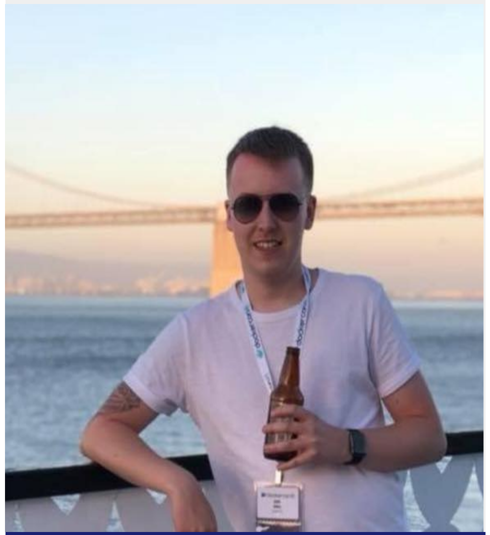
Dan Sheldon & Juan Herrera

Consulting Engineers - SUSE

Cloud_Native Rejekts [EU'22]

Intro

Introducing the speakers



Daniel Sheldon

Consulting Engineer

daniel.sheldon@suse.com

Juan Herrera Utande

Consulting Engineer

juan.herrera@suse.com

- SUSE EMEA Services Team
- Working on Kubernetes related projects using tools from Rancher’s ecosystem

Agenda

- Share our challenges in the field (we are consultants!)
- Review the use of Kubernetes in a non-canonical way
- Lessons learnt and discussions

Projects

Both projects see Kubernetes as an enabler technology but using nontraditional deployments.

5G Consortium

- Value added services on 5G towers (UPF, Edge Computing, …)
- Big rollout with huge cost implications
- New platform & deployment model (no existing references)

Industrial electronics company

- Add ”intelligence” to surveillance cameras
- ARM based architecture capable of running container workloads
- Mix on-premises with cloud services

Projects

Common denominator

- They don’t want remote locations to run full clusters. In the first project to reduce the HW needs and, on the second, due to the scarce resources available on the camera side
- Both see a value in Kubernetes, but their use cases are not “standard”

Remote worker nodes

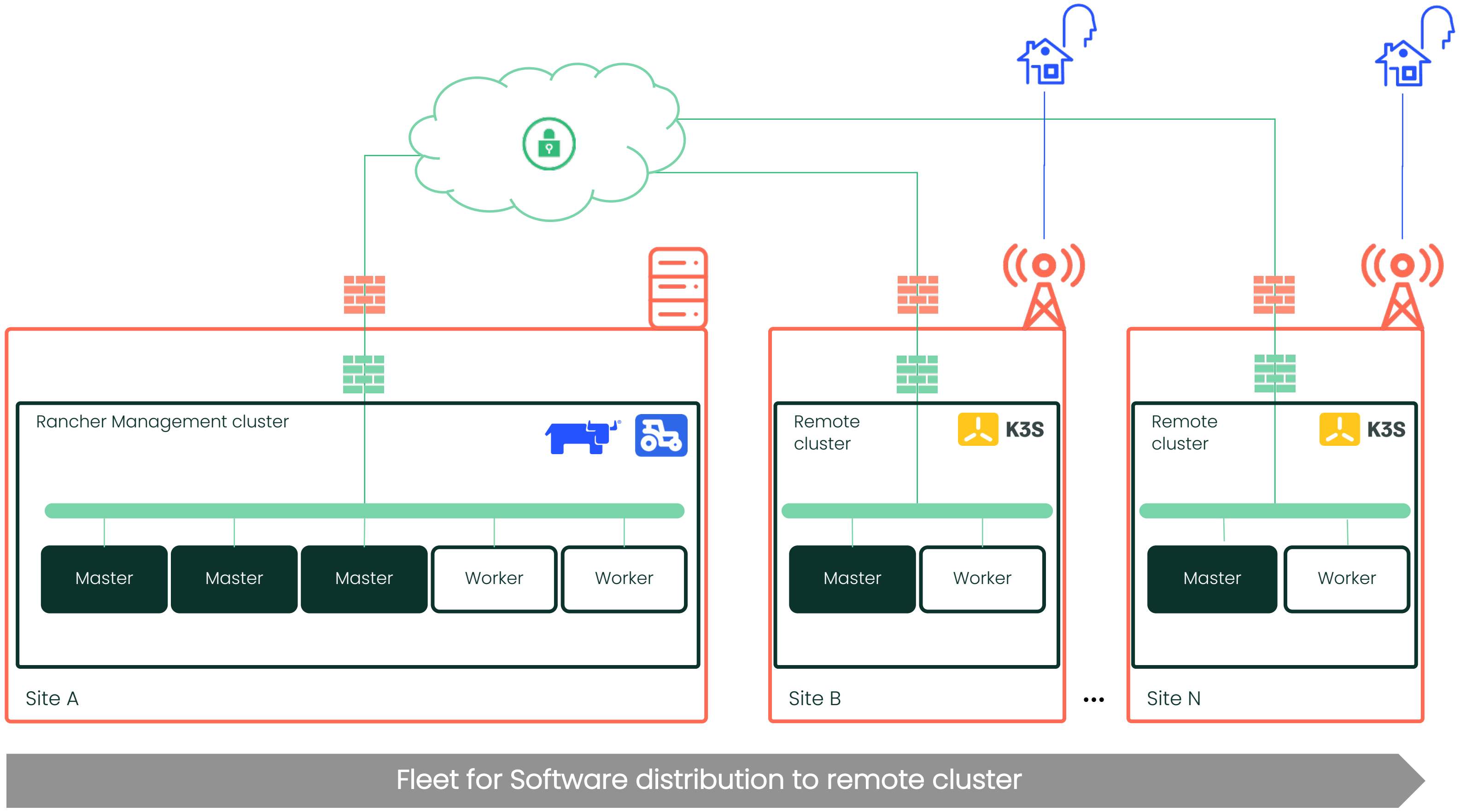
Common architectural requirement with the same challenges

- Challenges for Kubernetes scheduler (workload placement)
- Challenges for Kubernetes controller (keeping cluster healthy)
- Challenges for High Availability
- Challenges for Latency between sites
- Challenges for applications deployment

Worker nodes are going to be lonely out there and need to be self-sufficient …. (and that's an anti-best practice)

Remote Edge locations best practices

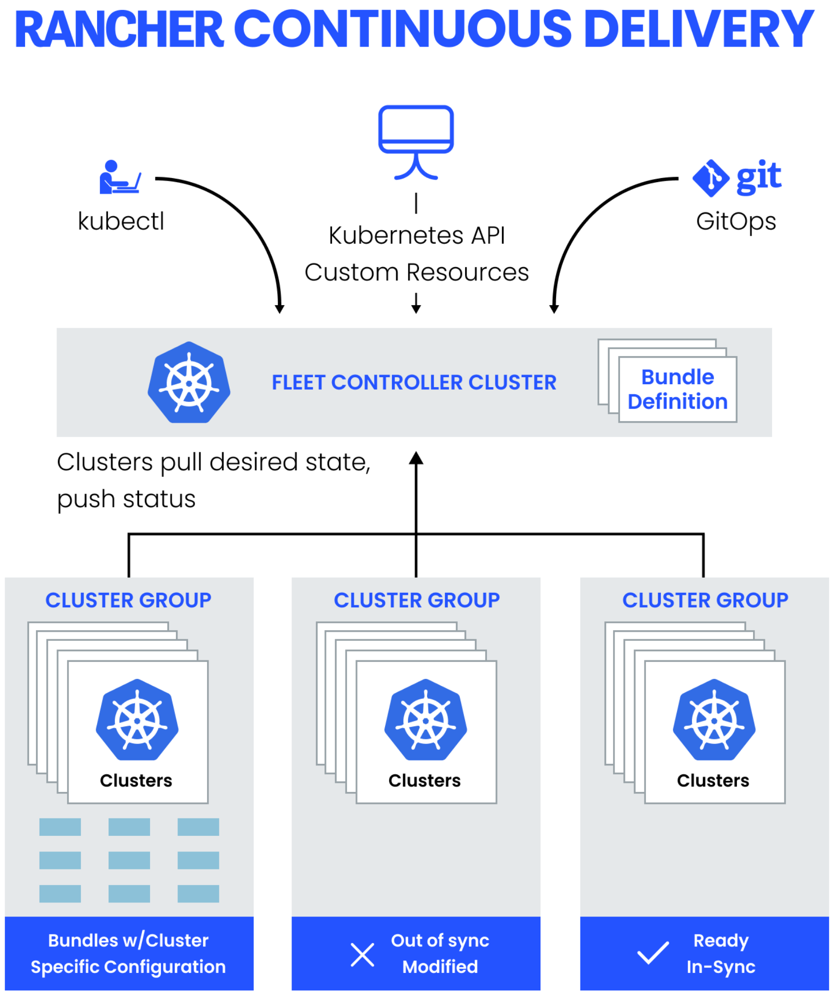
Which is our best practice for remote locations?



- ✓ Each location is a full cluster
- ✓ Resource constrained locations use single node with Master and Worker
- ✓ Rancher's Multi-cluster Management allows for a common pane of glass
- ✓ Centralized auth, management, software deployment, monitoring…

Why fleet for Edge application deployments?

- Rancher's foundation for its multi-cluster management features (well supported)
- Supports both k8s common deployments and Helm
- Fleet is a Kubernetes API extension. It can be managed through the REST API and deployment scenarios can be described as CRDs (GitOps ready)
- Unlimited scale (up to 1 million clusters)
- Works pretty well over unreliable networks (supports retries and handle temporary failures)
- Can even work in semi air-gapped environments (only small periods of connectivity)



Processes running on each node type

Master

- kube-apiserver
- etcd (*)
- kube-scheduler
- kube-controller-manager

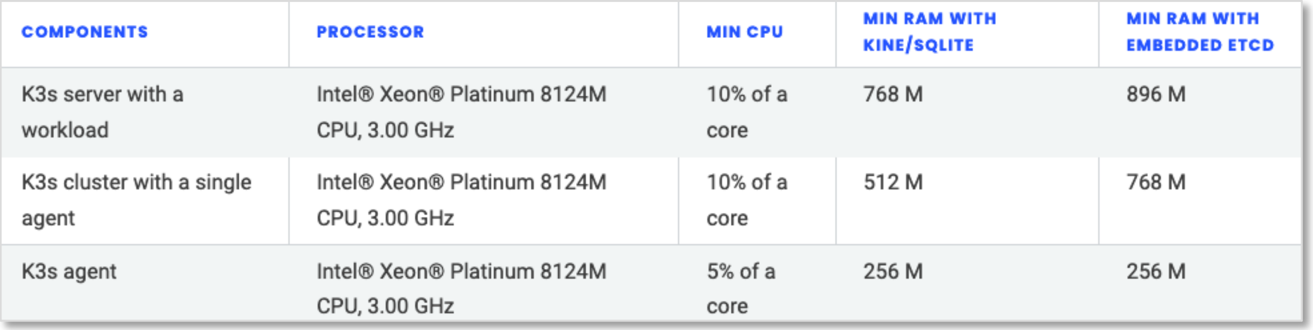
Worker

- kubelet
- kube-proxy
- Container runtime

Understanding k8s resources consumption

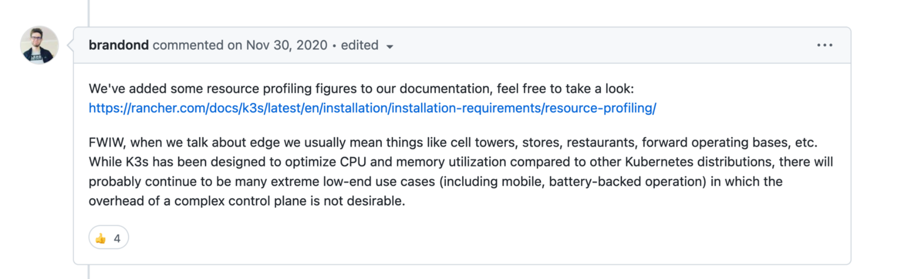
Reference: resource consumption analysis for k3s

https://rancher.com/docs/k3s/latest/en/installation/installation-requirements/resource-profiling/



COMPONENTS	PROCESSOR	MIN CPU	MIN RAM WITH KINE/SQLITE	MIN RAM WITH EMBEDDED ETCD
K3s server with a workload	Intel® Xeon® Platinum 8124M CPU, 3.00 GHz	10% of a core	768 M	896 M
K3s cluster with a single agent	Intel® Xeon® Platinum 8124M CPU, 3.00 GHz	10% of a core	512 M	768 M
K3s agent	Intel® Xeon® Platinum 8124M CPU, 3.00 GHz	5% of a core	256 M	256 M

A more in-depth discussion about Kubernetes resources consumption can be found on this GitHub thread: https://github.com/k3s-io/k3s/issues/2278



brandond commented on Nov 30, 2020 • edited

We've added some resource profiling figures to our documentation, feel free to take a look: https://rancher.com/docs/k3s/latest/en/installation/installation-requirements/resource-profiling/

FWIW, when we talk about edge we usually mean things like cell towers, stores, restaurants, forward operating bases, etc. While K3s has been designed to optimize CPU and memory utilization compared to other Kubernetes distributions, there will probably continue to be many extreme low-end use cases (including mobile, battery-backed operation) in which the overhead of a complex control plane is not desirable.

Remote worker node challenges

Remote Worker Nodes challenges

Relevant topics:

- Usually worker nodes do need “permanent” contact with the Master nodes
- Remote Worker Nodes may need custom configurations with unreliable links
- The amount of possible of failure scenarios increases
- How applications are deployment becomes relevant (not all models are valid)
- Network traffic considerations: east-west / north-south

Deployment options

- Static pod.
- DaemonSet:
  - Ok if node doesn’t reboot/boot while disconnected
  - Toleration for node failure scenarios: not-ready, unreachable,…
- Standard Deployments (Pod, Deployment, StatefulSet): with the right tolerations they can behave as DaemonSets. Also impacted by reboot/boot while disconnected.

Any deployment model should be driven by location related labels so workloads only live in specific nodes (worker nodes are being managed like Servers/VMs)

https://kubernetes.io/docs/concepts/workloads/controllers

The Reboot/boot problem

If a worker node is restarted while the connection to the control plane is not available, Pods will not start even if the right tolerations are in place.

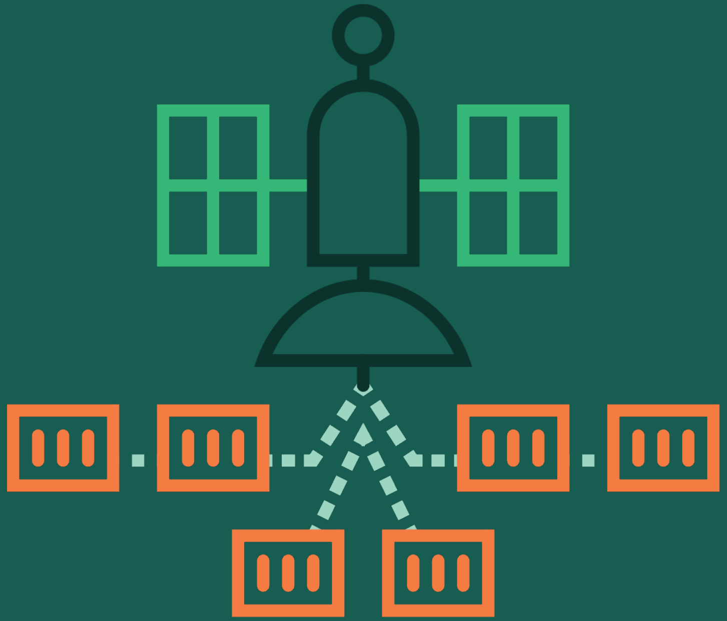
The only pods that work in this scenario are static pods.

Deployment of workloads using static Pods can be challenging. Kubelet process in worker nodes can pull Pod definition from a URL. Kubelet parameters:

- --manifest-url string (deprecated)
- staticPodURL (kubelet config file parameter)

Telco

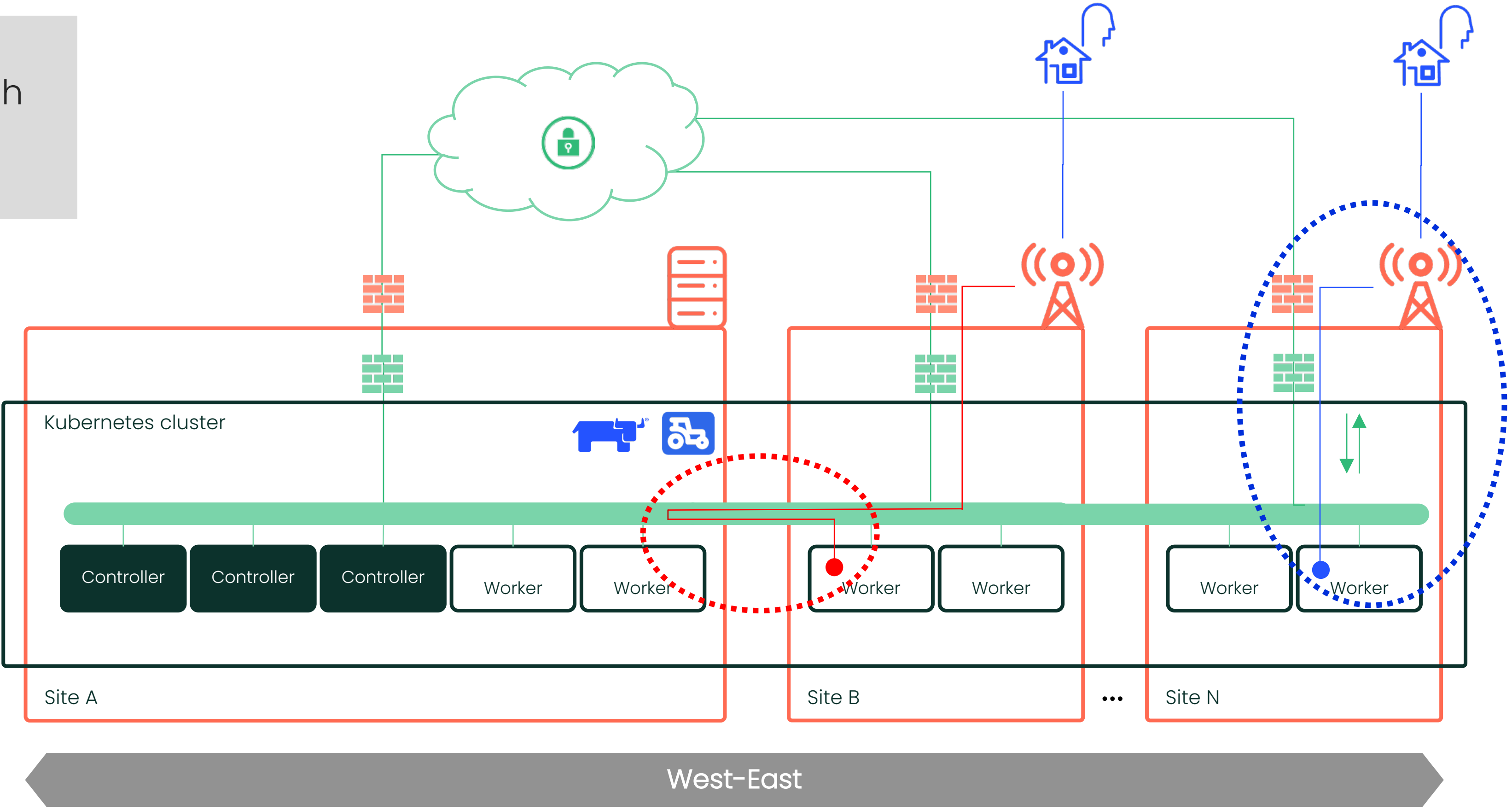
Remote Worker Nodes in a Telco 5G environment



Telco: Networking

Challenge: keep network traffic local per location

- Blue path is the right approach
- Red path should be avoided



Challenge: keep network traffic local per location

East-West

- The only east-west traffic that should exist is the Master connecting to the Workers for cluster health checks and application deployments.

East-West/North-South traffic is impacted by how applications are made visible

- NodePorts and ClusterIP Service types have cluster wide visibility and live in the shared overlay network. There are placement rules for workloads but not for services. Traffic “may” cross site boundaries.
- HostPort is the only model that guarantees that the workload is only accessed locally
- Full local traffice can also be achieved with Multus if dedicated network cards or SRIOV capable network cards available

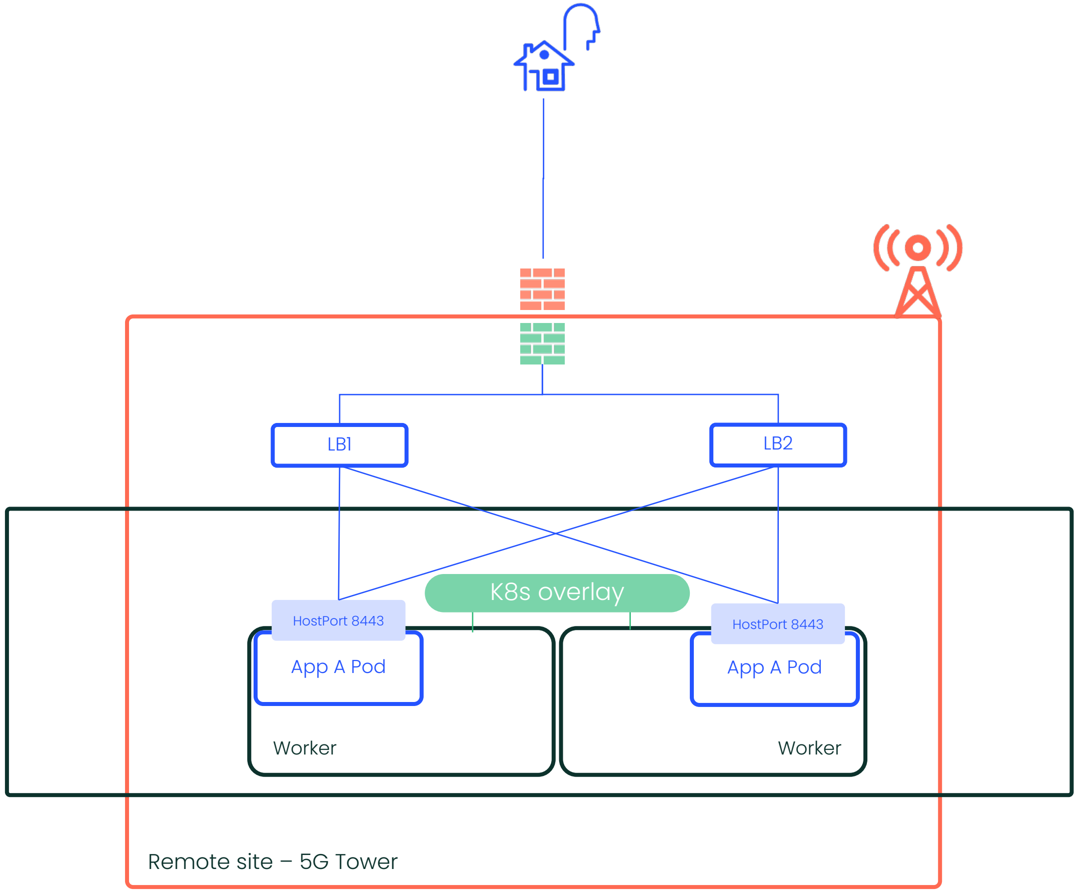
Other topics

- Use of local Ingress on remote locations (global ingress should be avoided)

Example: Local app using HostPath

Comments

- Simplest model
- Basic HA
- No need to create services
- Users manually control Port assignments and LB rules

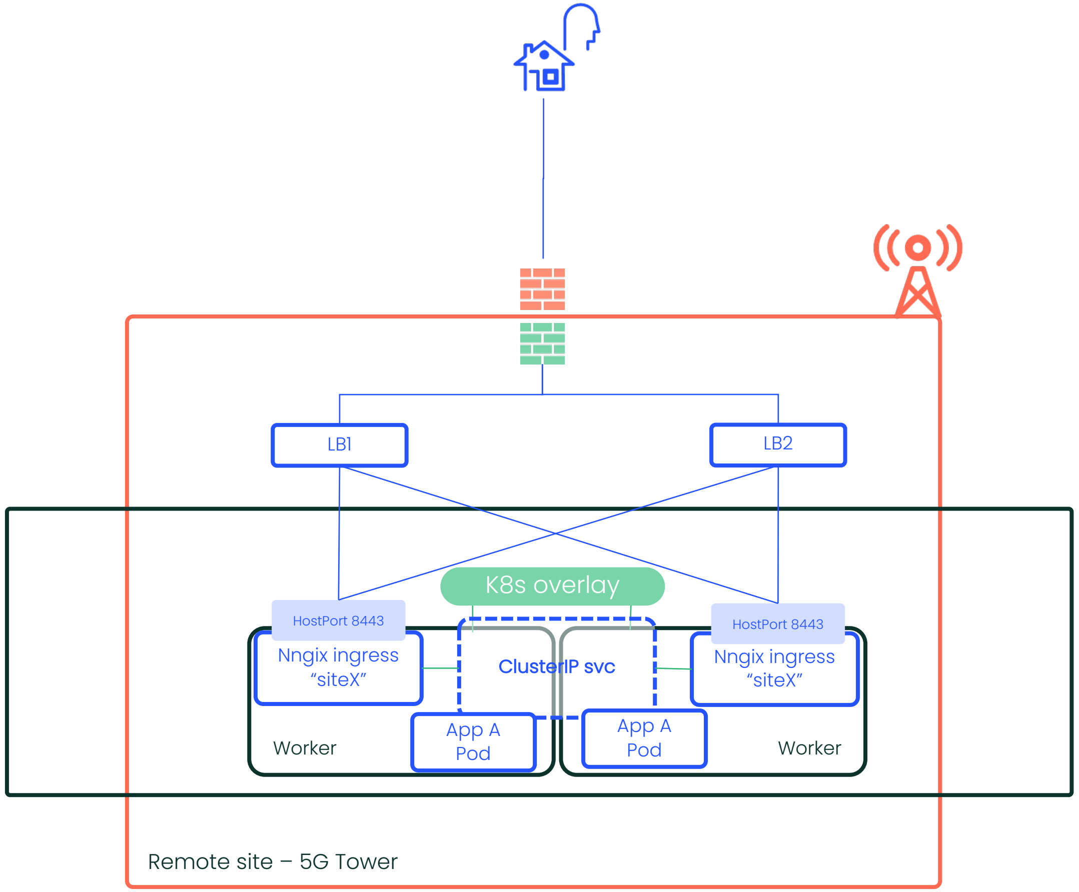


Cloud_Native Rejekts [EU'22]

Example: Local Ingress + HostPort + Load Balancer

Comments

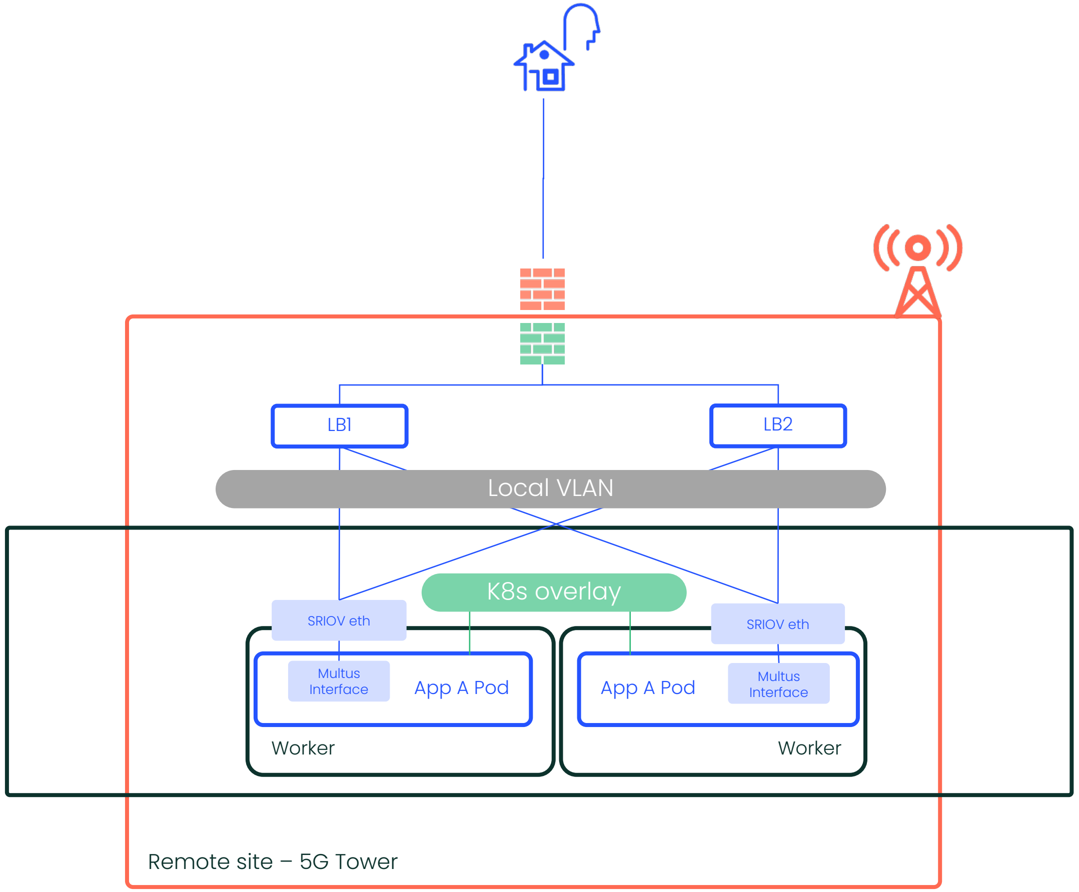
- Only Ingress itself needs to use HostPort, that will keep app deployments cleaner
- Ingress is labeled and deployed only on the desired remote location
- No global Ingress needed. Each location will have their own Ingress pods
- CNI plugin will optimize traffic so traffic to and from cluster IP will remain local*



Chosen solution

Multus + SR-IOV + local VLAN

- Full guarantee traffic will be local
- Skipping k8s overlay network will improve network performance
- No need to create services
- No need to use HostPort
- Users manually control Port assignments and LB rules



Telco: Common Configurations

Cluster, node and workload configurations

Scenarios with remote worker nodes exposed to unexpected latencies or network outages may benefit from tweaking some configurations (but they are optional)

Three types of configurations:

- **Cluster wide**: changes to kube-controller and kube-api
- **Node**: changes to kubelet config
- **Workload**: taints and tolerations

Summary of configuration changes

Global and Node level

```
services:
 kubelet:
   extra_args:
     node-status-update-frequency: 4s > 8s
 kube-api:
   extra_args:
     default-not-ready-toleration-seconds: 30 > 60
     default-unreachable-toleration-seconds: 30 > 60
 kube-controller:
   extra_args:
     node-monitor-period: 2s > 4s
     node-monitor-grace-period: 16s > 32s
     pod-eviction-timeout: 30s > 60s
```

All changes can be added to the cluster.yaml configuration file on Rancher - RKE clusters

Summary of configuration changes

Workloads

Workers losing contact with Masters will evict running pods unless those scenarios are explicitly tolerated

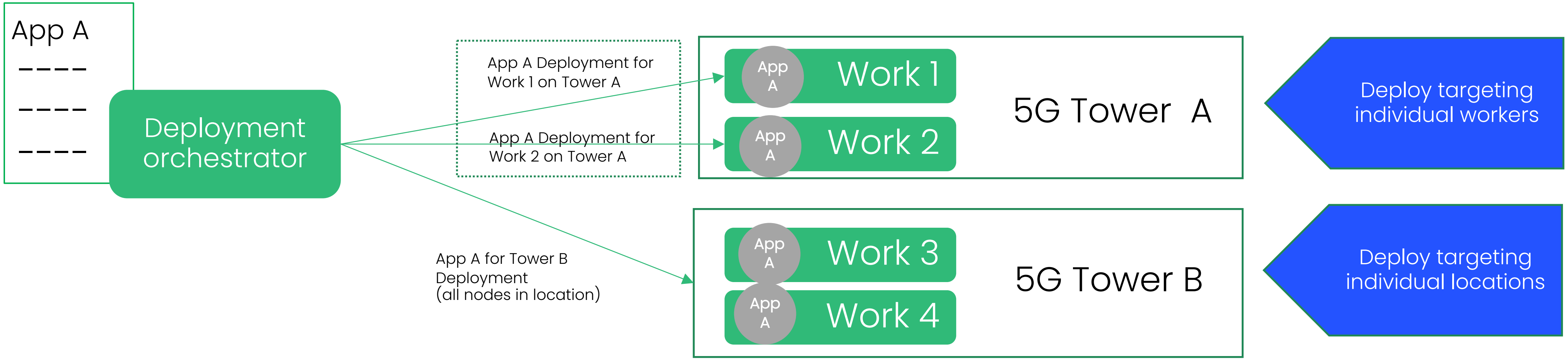
```
tolerations:
  - key: node.kubernetes.io/unreachable
    operator: Exists
    effect: NoExecute
    #tolerationSeconds: 21600
  - key: node.kubernetes.io/not-ready
    operator: Exists
    effect: NoExecute
    #tolerationSeconds: 21600
  - key: node.kubernetes.io/unschedulable
    operator: Exists
    effect: NoSchedule
    #tolerationSeconds: 21600
```

(DaemonSets already include those tolerations)
(Tolerations won't work if the node is rebooted/started will disconnected from master)

Telco: Application deployment

Application deployment

- Managed by an external orchestrator (ONAP)
- Each App will have as many deployment as target 5G towers
- Deployments will target 5G towers using node selectors
- Node labels will be used to uniquely identify both locations and workers. VM "like" deployment
- Labeling of Ingress, Services, Volumes, ... is also highly recommended



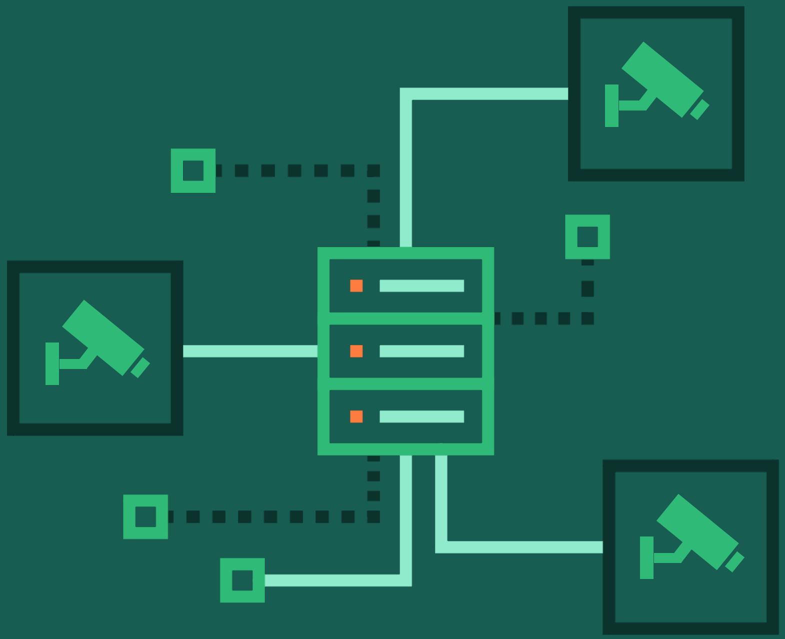
App A

Deployment orchestrator
App A Deployment for Work 1 on Tower A
App A Deployment for Work 2 on Tower A
App A for Tower B Deployment (all nodes in location)
App A
Work 1
App A
Work 2
5G Tower A
App A
Work 3
App A
Work 4
5G Tower B
Deploy targeting individual workers
Deploy targeting individual locations

Cloud_Native Rejekts [EU'22]

Cameras

Containerized AI monitoring system

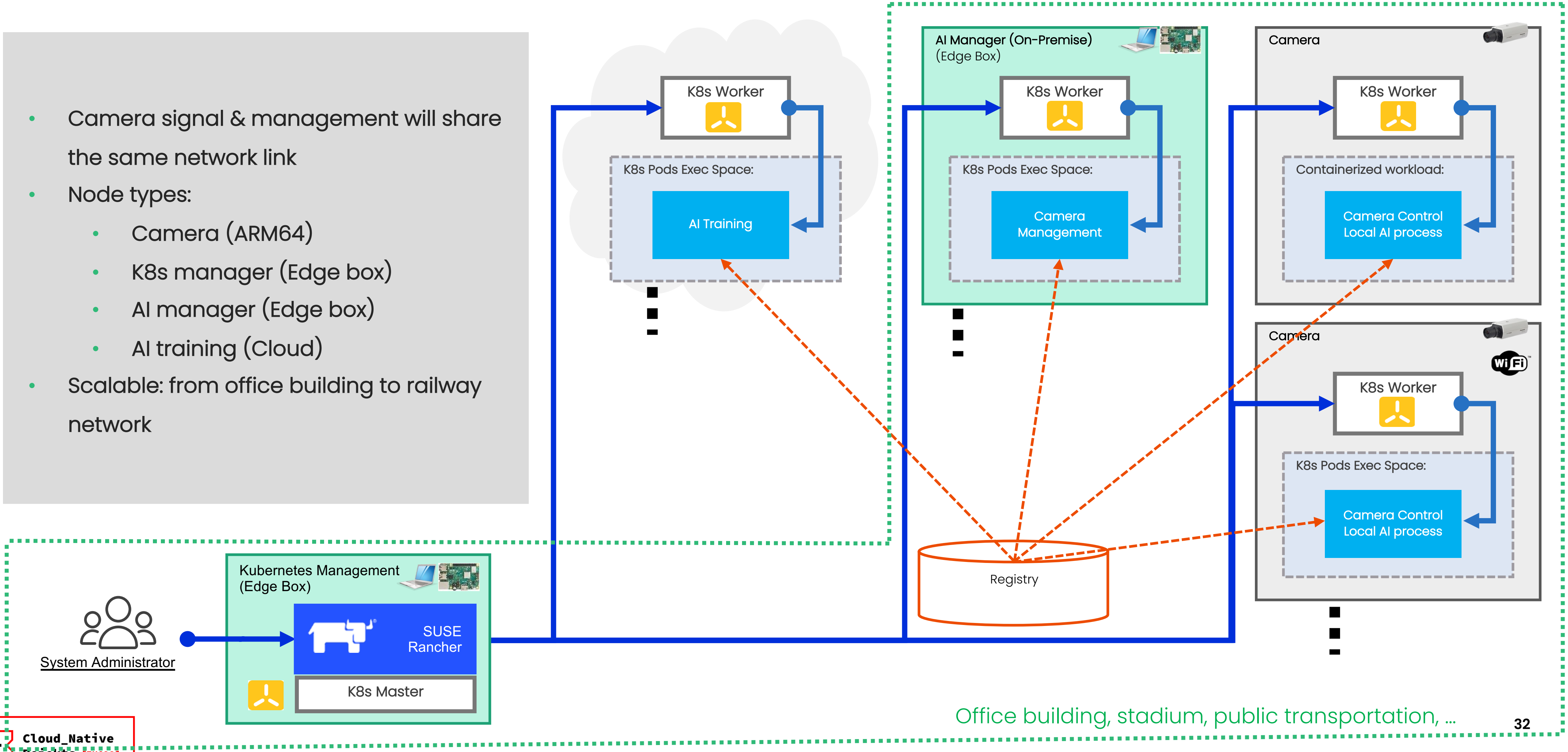


Cloud_Native
Rejekts [EU'22]

Cameras: Networking

Challenge: shared WIFI links / future scale

- Camera signal & management will share the same network link
- Node types:
  - Camera (ARM64)
  - K8s manager (Edge box)
  - AI manager (Edge box)
  - AI training (Cloud)
- Scalable: from office building to railway network



Office building, stadium, public transportation, ...

Cameras: Common Configurations

Summary of configuration changes

Global and Node parameters level will keep the default values

```
services:
 kubelet:
   extra_args:
     node-status-update-frequency: 4s
 kube-api:
   extra_args:
     default-not-ready-toleration-seconds: 30
     default-unreachable-toleration-seconds: 30
 kube-controller:
   extra_args:
     node-monitor-period: 2s
     node-monitor-grace-period: 16s
     pod-eviction-timeout: 30s
```

All changes can be added to the cluster.yaml configuration file on Rancher - RKE clusters

Summary of configuration changes

Workloads

Edge box services will use standard deployments
Camera deployments will be based on DaemonSets so default tolerations will be used:

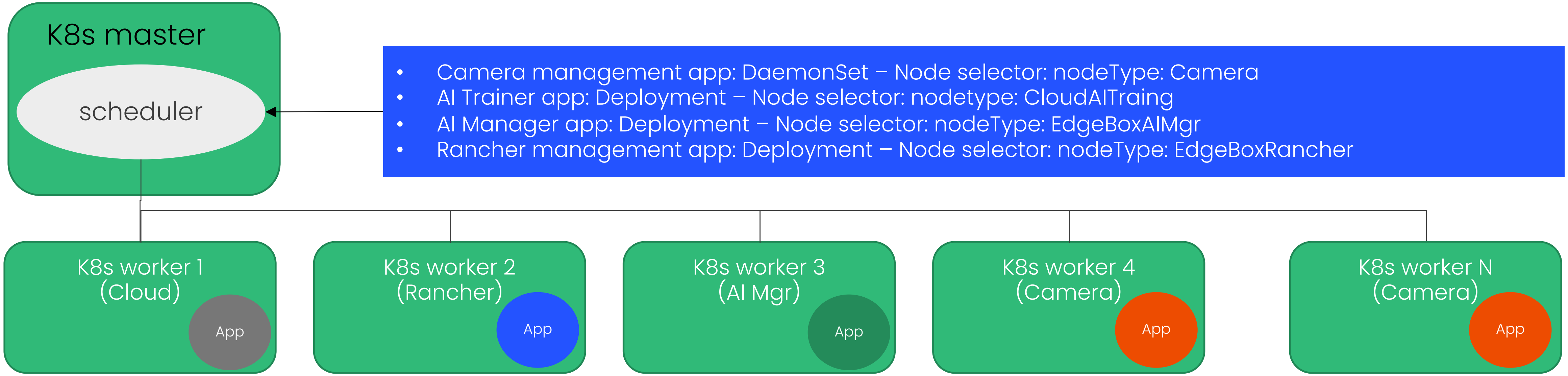
```
toleration s:
```

(DaemonSets already include those tolerations but explicitly declaring them is recommended)
(Tolerations won't work if the node is rebooted/started will disconnected from master)

Cameras : Application deployment

Application deployment

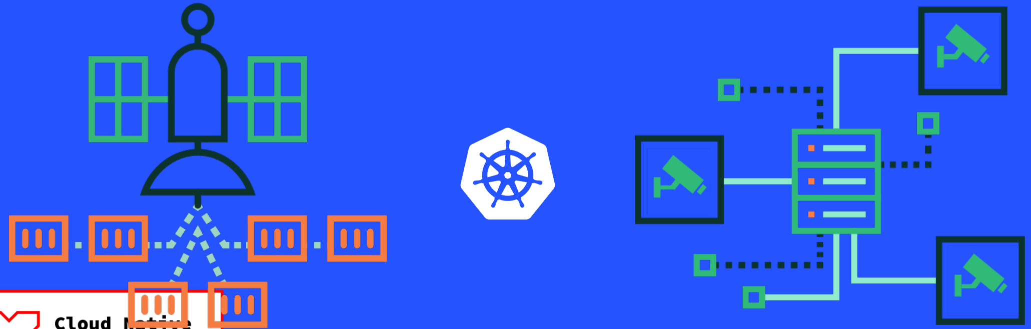
- No external orchestrator. Just standard Kubernetes objects
- Using of DaemonSets and NodeSelector for cameras. Standard Deployments for the rest
- Each camera worker will have two labels: cameraID:camera_SN and nodeType: camera
- Camera applications will be deployed as DaemonSet with node selector using nodeType: Camera
- AI training, Rancher and Camera Management target through node selectors



- Camera management app: DaemonSet – Node selector: nodeType: Camera
- AI Trainer app: Deployment – Node selector: nodetype: CloudAITraing
- AI Manager app: Deployment – Node selector: nodeType: EdgeBoxAIMgr
- Rancher management app: Deployment – Node selector: nodeType: EdgeBoxRancher

Q & A

Thank you!



Cloud_Native Rejekts [EU'22]