SDC 19

September 23-26, 2019
Santa Clara, CA

MetalK8s

An opinionated Kubernetes distribution optimized for data management

Nicolas Trangez
Senior Architect, Scality
@eikke | @scality | @zenko

ABOUT SCALITY

GLOBAL PRESENCE

8 OFFICES

200+ PEOPLE

20+ NATIONALITIES

GLOBAL CLIENT BASE

EUROPE

120+

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE DE LA DÉFENSE
orange
METROPOLITAN POLICE
MAIRIE DE PARIS
RTL
SFR
EUROSPORT
COM HEM
dailymotion
NATIXIS
Molotov.tv
TF1
Hospices Civils de Lyon
RENAULT
SNCF
AB
SOCIETE GENERALE
deluxe
EDF
AIRBUS

AMERICAS

60+

Penn Medicine
Bank of America
Bloomberg Media
COMCAST
STAPLES
logitech
HBO
rackspace

AUSTRALIA

~10

EndemolShine Group
MediaHub
foxtel

JAPAN

20+

KDDI
BIGLOBE
DMM.com
@nifty

SCALITY

OUR JOURNEY TO KUBERNETES

Scality RING, S3 Connector & Zenko

SCALITY

Scality RING

On-premise
Distributed Object & File Storage

- Physical servers, some VMs
- Only the OS available (incl. ‘legacy’ like CentOS 6)
- Static resource pools
- Static server roles / configurations
- Solution distributed as RPM packages, deployed using SaltStack
- De-facto taking ownership of host, difficult to run multiple instances
- Fairly static post-install

SCALITY

Scality S3 Connector

On-premise S3-compatible Object Storage

- Physical servers, sometimes VMs
- Static resource pools
- “Microservices” architecture
- Solution distributed as Docker container images, deployed using Ansible playbooks
- No runtime orchestration
- Log management, monitoring,... comes with solution

SCALITY

Scality Zenko

Multi-Cloud Data Controller

- Deployed on-prem or ‘in the Cloud’: major paradigm shift
- New challenges, new opportunities
- Multi-Cloud Data Controller, must run on multiple Cloud platforms
- See Vianney’s talk!

SCALITY

Scality Zenko

Deployment Model

- Embraced containers as distribution mechanism
  - Some shared with Scality S3 Connector
- Decided to move to Kubernetes
  - Managed platforms for Cloud deployments, where available (GKE, AKS, EKS,...)
  - On-prem clusters

SCALITY

Scality Zenko

Kubernetes Benefits

- Homogenous deployment between in-cloud and on-prem
- Various services provided by cluster:
  - Networking & policies
  - Service restart, rolling upgrades
  - Service log capturing & storage
  - Service monitoring & metering
  - Load-balancing
  - TLS termination
- Flexible resource management
  - If needed, easily add resources to cluster by adding some (VM) nodes
  - HorizontalPodAutoscaler

SCALITY

Scality Zenko

Kubernetes Deployment

- Currently using Helm chart
- Contributed many fixes and features to upstream charts repository
- Contributed new charts and became maintainer of some others
- Next-gen: Zenko ‘operator’
- Can run in your cluster (https://github.com/Scality/Zenko) or test-drive a hosted instance for free using Zenko Orbit at https://zenko.io/admin

INTERLUDE

What is Kubernetes?

Kubernetes

- “Container orchestration system”
- Actually:
  - Extensible model of ‘desired reality’ in a distributed database, accessible through RESTful APIs
  - Authn / authz
  - Controllers to refine high-level object(s) into lower-level ones and/or reconcile reality with model
  - Service(s) running on machines to realize desired reality at system level
  - Set of APIs/plugins to interact with system: CRI, CNI, CSI
- In essence, ‘containers’ are only a small part of the K8s architecture
- Core unit of scheduling: Pod
  - Set of containers running in single network namespace
  - Pods can request attachment to persistent storage volumes

OUR JOURNEY TO KUBERNETES

MetalK8s

SCALITY

On-prem Kubernetes

- Can’t expect a Kubernetes cluster to be available, provided by Scality customer
- Looked into various existing offerings, but in the ends needs to be supported by/through Scality (single offering)
    - Also, many existing solutions don’t cover enterprise datacenter requirements
- Decided to roll our own

SCALITY METALK8S

AN OPINIONATED KUBERNETES DISTRIBUTION WITH A FOCUS ON LONG-TERM ON-PREM DEPLOYMENTS

OPINIONATED

We offer an out-of-the-box experience, no non-trivial choices to be made by users

LONG-TERM

Storage is mission-critical and *sticky*, can’t spawn a new cluster to upgrade and use a load-balancer in front

ON-PREM

Can’t expect anything to be available but (physical) servers with a base OS

MetalK8s 1.x

- Scope: 3-20 physical machine, pre-provisioned by customer or partner
- Built on top of the Kubespray Ansible playbook
- Use Kubespray to lay out a base Kubernetes cluster
  - Also: etcd, CNI
- Add static & dynamic inventory validation pre-checks, OS tuning, OS security
  - Based on experience from large-scale Scality RING deployments
- Augment with various services, deployed using Helm
  - Operations
  - Ingress
  - Cluster services
- Take care of on-prem specific storage architecture

MetalK8s 1.x: Cluster Services

- “Stand on the shoulders of giants”
- Heapster for dashboard graphs, `kubectl top`,...
- metrics-server for HorizontalPodAutoscaler
  - Looking into k8s-prometheus-adapter
- Ingress & TLS termination: nginx-ingress-controller
- Cluster monitoring & alerting: Prometheus, prometheus-operator, Alertmanager, kube-prometheus, Grafana
  - Host-based node_exporter on all servers comprising the cluster, including etcd
- Host & container logs: Elasticsearch, Curator, fluentd, fluent-bit, Kibana
- All of the above gives a great out-of-the-box experience for operators

kubernetes

Search

+ CREATE

Overview

Cluster

- Namespaces
- Nodes
- Persistent Volumes
- Roles
- Storage Classes

Namespace

zenko

Overview

Workloads

- Cron Jobs
- Daemon Sets
- Deployments
- Jobs
- Pods
- Replica Sets
- Replication Controllers
- Stateful Sets

Discovery and Load Balancing

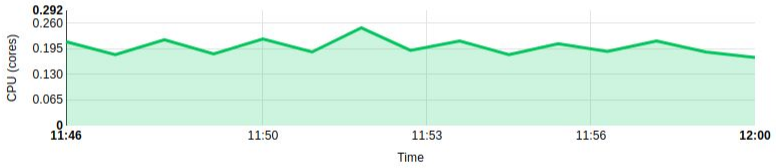
- Ingresses
- Services

Config and Storage

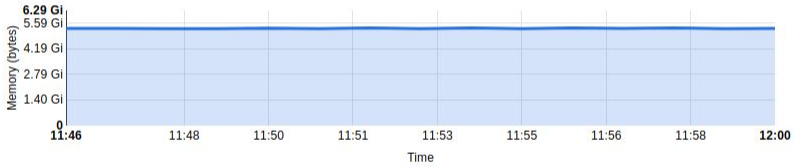
- Config Maps
- Persistent Volume Claims
- Secrets

Settings

CPU usage



Memory usage



Workloads

Workloads Statuses

Deployments

Pods

Replica Sets

Stateful Sets

Deployments

Name	Labels	Pods	Age	Images
zenko1-backbeat-consumer	app: backbeat-consumer chart: backbeat-consumer-0.1.0 heritage: Tiller release: zenko1	1 / 1	2 days	zenko/backbeat:0.1.4
zenko1-backbeat-producer	app: backbeat-producer chart: backbeat-producer-0.1.0 heritage: Tiller release: zenko1	1 / 1	2 days	zenko/backbeat:0.1.4
zenko1-cloudserver-front	app: cloudserver-front chart: cloudserver-front-0.1.3 heritage: Tiller release: zenko1	1 / 1	2 days	zenko/cloudserver:0.1.6

kibana

4,743 hits

New Save Open Share Auto-refresh Last 15 minutes

Search... (e.g. status:200 AND extension:PHP) Uses lucene query syntax

- Discover
- Visualize
- Dashboard
- Timelion
- Dev Tools
- Management

kubernetes.labels.release: "zenko1" Add a filter +

Actions

logstash-*

Selected Fields

- t kubernetes.host
- t kubernetes.labels.app
- t log

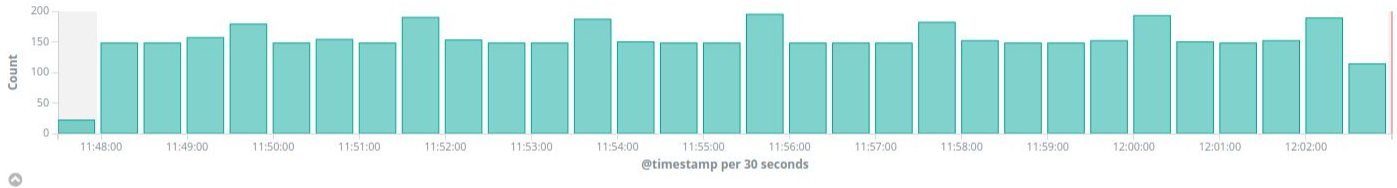
Available Fields

Popular

- t kubernetes.container_name
- @timestamp
- t _id
- t _index
- # _score
- t _type
- ? address
- ? clientIP
- ? clientPort
- ? configurationVersion
- t docker.container_id
- ? error.address
- ? error.code
- ? error.errno
- ? error.port
- ? error.syscall
- ? hostname
- ? httpMethod
- ? httpURL
- ? https
- t kubernetes.labels.chart
- t kubernetes.labels.controller-revisio...

Collapse

March 26th 2018, 11:47:57.351 - March 26th 2018, 12:02:57.351 — Auto

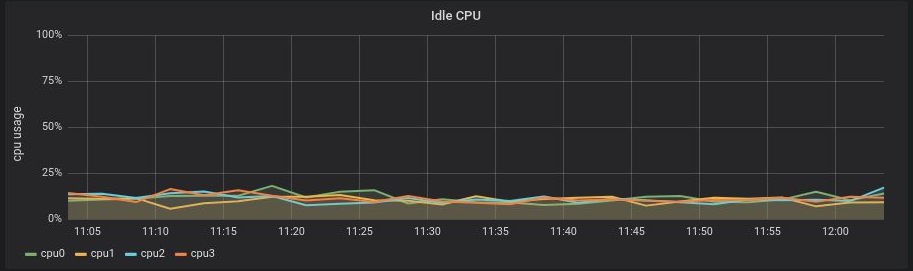


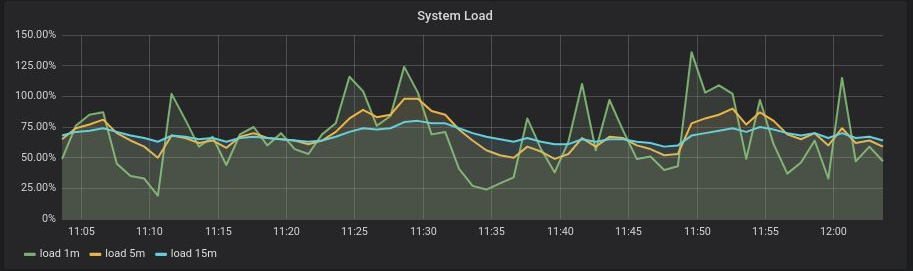
Time	kubernetes.labels.app	kubernetes.host	log
March 26th 2018, 12:02:53.930	mongodb-replicaset	metalk8s5-03	2018-03-26T19:02:53.930+0000 I - [conn22297] end connection 127.0.0.1:58200 (15 connections now open)
March 26th 2018, 12:02:53.928	mongodb-replicaset	metalk8s5-03	2018-03-26T19:02:53.928+0000 I NETWORK [thread1] connection accepted from 127.0.0.1:58200 #22297 (15 connections now open)
March 26th 2018, 12:02:53.928	mongodb-replicaset	metalk8s5-03	2018-03-26T19:02:53.928+0000 I NETWORK [conn22297] received client metadata from 127.0.0.1:58200 conn22297: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "3.4.14" }, os: { type: "Linux", name: "PRETTY_NAME="Debian GNU/Linux 8 (jessie)"", architecture: "x86_64", version: "Kernel 4.4.0-116-generic" } }
March 26th 2018, 12:02:53.280	mongodb-replicaset	metalk8s5-04	2018-03-26T19:02:53.280+0000 I - [conn20573] end connection 127.0.0.1:41600 (13 connections now open)
March 26th 2018, 12:02:53.278	mongodb-replicaset	metalk8s5-04	2018-03-26T19:02:53.277+0000 I NETWORK [conn20573] received client metadata from 127.0.0.1:41600 conn20573: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "3.4.14" }, os: { type: "Linux", name: "PRETTY_NAME="Debian GNU/Linux 8 (jessie)"", architecture: "x86_64", version: "Kernel 4.4.0-116-generic" } }
March 26th 2018, 12:02:53.277	mongodb-replicaset	metalk8s5-04	2018-03-26T19:02:53.277+0000 I NETWORK [thread1] connection accepted from 127.0.0.1:41600 #20573 (13 connections now open)
March 26th 2018, 12:02:50.643	mongodb-replicaset	metalk8s5-04	2018-03-26T19:02:50.643+0000 I - [conn20572] end connection 127.0.0.1:41594 (13 connections now open)
March 26th 2018, 12:02:50.638	mongodb-replicaset	metalk8s5-04	2018-03-26T19:02:50.638+0000 I NETWORK [conn20572] received client metadata from 127.0.0.1:41594 conn20572: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "3.4.14" }, os: { type: "Linux", name: "PRETTY_NAME="Debian GNU/Linux 8 (jessie)"", architecture: "x86_64", version: "Kernel 4.4.0-116-generic" } }
March 26th 2018, 12:02:50.638	mongodb-replicaset	metalk8s5-04	2018-03-26T19:02:50.638+0000 I NETWORK [thread1] connection accepted from 127.0.0.1:41594 #20572 (13 connections now open)
March 26th 2018, 12:02:50.521	mongodb-replicaset	metalk8s5-01	2018-03-26T19:02:50.521+0000 I - [conn21254] end connection 127.0.0.1:45634 (9 connections now open)
March 26th 2018, 12:02:50.518	mongodb-replicaset	metalk8s5-01	2018-03-26T19:02:50.517+0000 I NETWORK [conn21254] received client metadata from 127.0.0.1:45634 conn21254: { application: { name: "MongoDB Shell" }, driver: { name: "MongoDB Internal Client", version: "3.4.14" }, os: { type: "Linux", name: "PRETTY_NAME="Debian GNU/Linux 8 (jessie)"", architecture: "x86_64", version: "Kernel 4.4.0-116-generic" } }

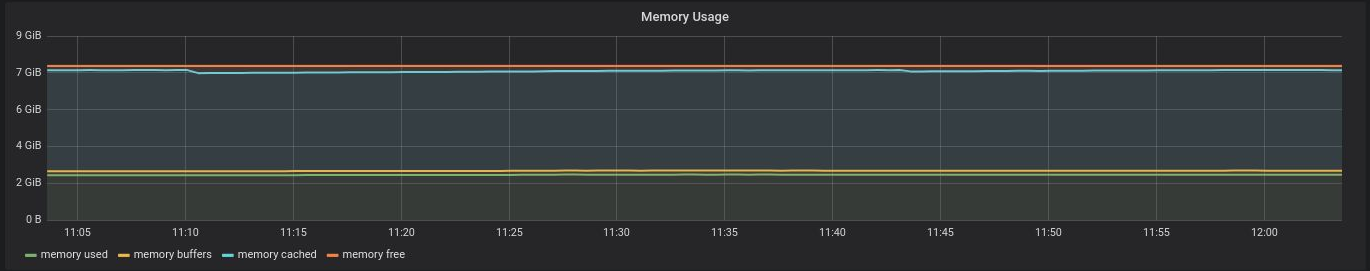
Nodes

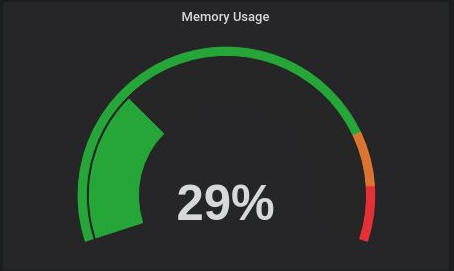
Last 1 hour

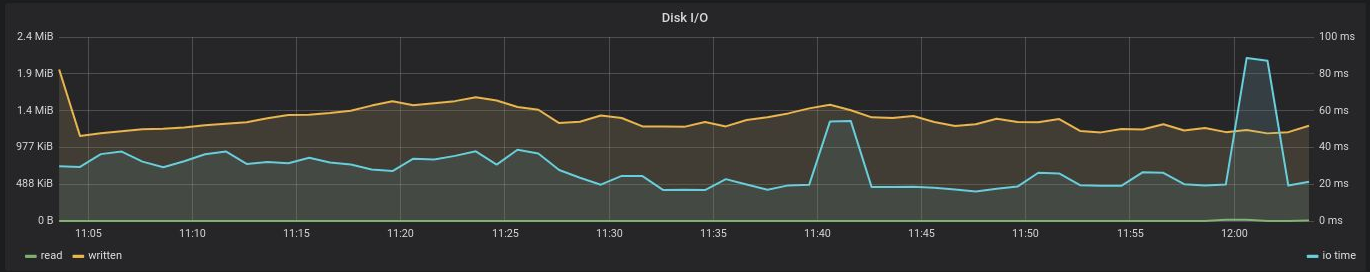
server 10.100.2.227:9100

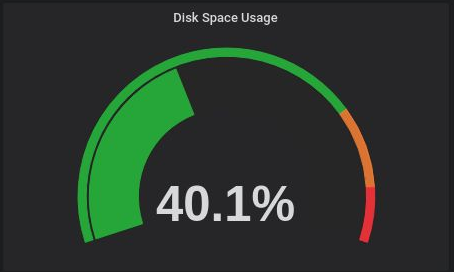


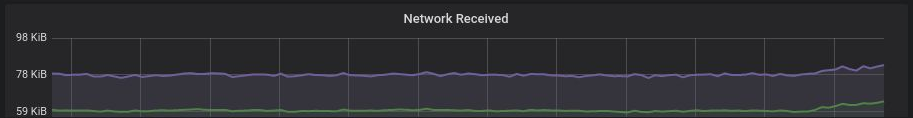


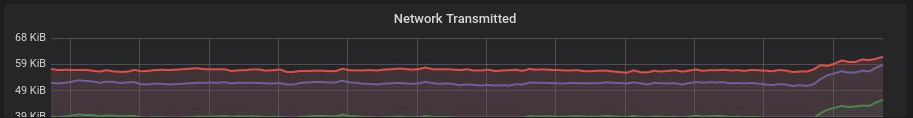












INTERLUDE

Storage in a Kubernetes Cluster

Storage in Kubernetes

- Not tied to particular implementation
  - File and block supported
  - Local disks/file systems or attached SAN/NAS
- Data model
  - StorageClass (SC): primarily used to configure on-demand provisioning
  - PersistentVolume (PV): representation of an available volume
  - PersistentVolumeClaim (PVC): binding between a Pod and a PV
- Classic on-demand provisioning:
  - Create PVC for given SC
  - Provisioner configured in SC creates backing storage and creates PV
  - PVC bound to PV
- CSI plugins perform attach and mounting of volumes
  - Currently many built-in, being split out of core

MetalK8s 1.x: Storage

- On-prem: no EBS, no GCP Persistent Disks, no Azure Storage Disk,...
- Also: can't rely on NAS (e.g. through OpenStack Cinder) to be available
- Lowest common denominator: local disks in a node
- Decided to use static provisioning during installation
  - Based on LVM2 Logical Volumes for flexibility
  - PV, VG, LVs defined in inventory, created/formatted/mounted by playbook
  - K8s PV objects created by playbook

Roadblocks and new requirements

- Based on years of years of experience deploying Scality RING at enterprise customers, service providers,...
- Constraints in data centers often very different from 'VMs on EC2'
  - No direct internet access: everything through HTTP(S) proxy, no non-HTTP traffic
  - Security rules requiring services to bind to specific IPs only, different subnets for control & workload,...
  - Fully air gapped systems: requires 100% offline installation
  - Non-standard OS/kernel
  - Integration with corporate authn/authz systems
  - Loosely captured in https://github.com/scality/metalk8s/blob/development/2.0/docs/requirements.rst
- Given reqs, decided continuing same path wouldn't work
  - However, useful lessons learned

Scality MetalK8s 2.x

Platform for next-gen products

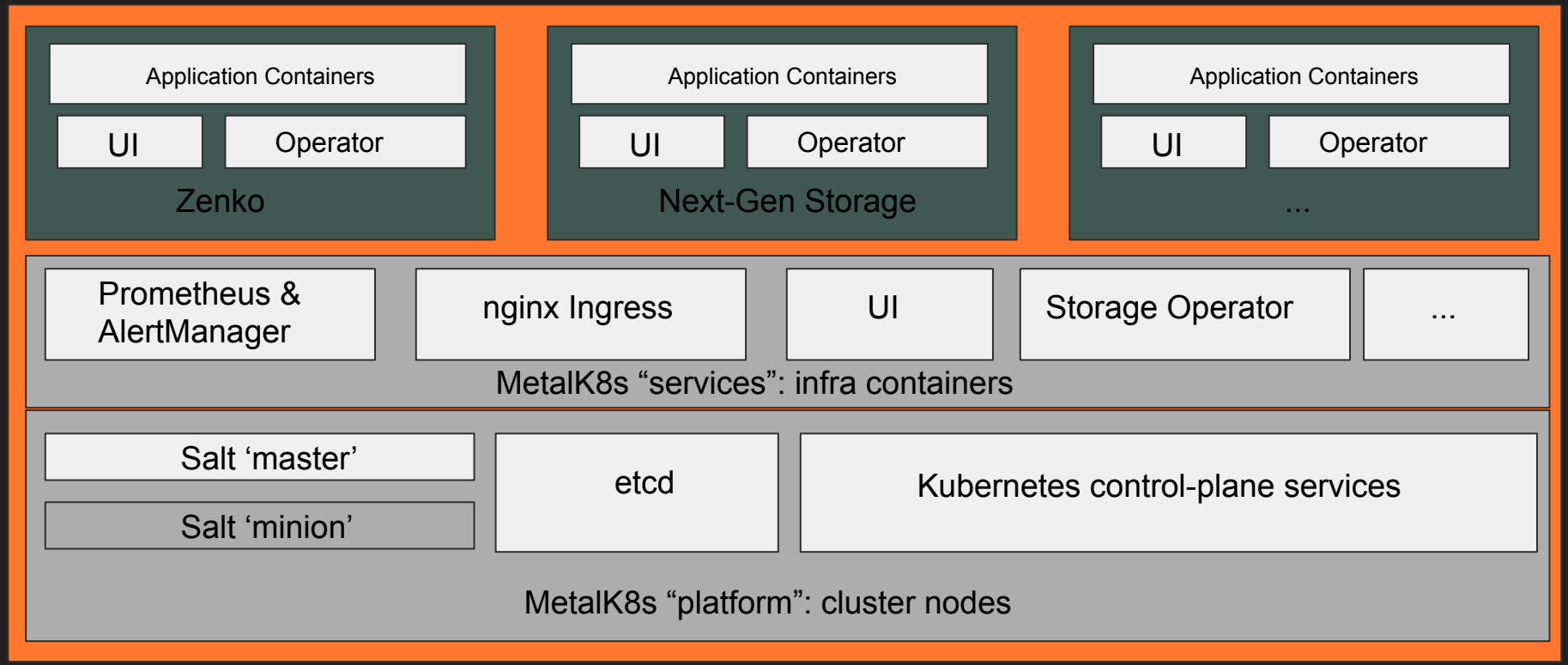
SCALITY

MetalK8s: Shifting focus

- 1.x: general-purpose deployment tool, fulfil K8s cluster pre-req of $product

- 2.x: use-case specific component a vendor (you!) can embed in on-prem solution/product running on K8s without being a K8s product
  - More configurable to match exact solution requirements and deployment environment
  - Tighten out-of-the-box security depending on application ‘insecurity’ needs

MetalK8s 2.x: Changed Architecture



SCALITY

MetalK8s 2.x: Changed Architecture

- Not built on Kubespray, built from scratch, isomorphic to *kubeadm* cluster
- No one-of playbooks, no ‘deploy-only’ clusters: focus on day 1/N operations
- Need a dynamic system:
  - React to potential issues in cluster (e.g. prevent certificate TTL expiration)
- Need to act on system layer ‘from inside K8s’
  - Cluster expansion
  - Storage provision
  - Cluster upgrade/downgrade

MetalK8s 2.x: Changed Architecture

- Using SaltStack
  - Dynamic, ‘always-present’, lots of in-house experience
- Build everything into single ISO image
- Cluster deployment / lifecycle:
  - Deploy ‘bootstrap’ node: mount ISO, create minimal config, run script
  - Deployment kickstarts from Bash into Salt local mode, then master/minion mode ASAP
  - Lays out Yum repository, container registry, Salt ‘master’ on bootstrap node, 1-node K8s cluster
  - CRI: *containerd*, no Docker. Maybe *cri-o* someday
  - CNI: Calico
  - Once bootstrap deployed to fully-functional 1-node K8s cluster, add new nodes:
    - Create *Node* object in K8s, set *role* labels and SSH info
    - Trigger Salt to deploy server: orchestration uses SaltSSH for minion, then plain Salt
    - Through API, CLI or UI
    - Control-plane, infra, workload node, or mixture
  - Cluster upgrade and downgrade fully automated, rolling

Interlude: static-container-registry

- Used to deploy full Docker Registry as part of MetalK8s
- Overkill: only need read-only distribution
- Overkill: requires ‘seeding’ the registry during deployment
- Overkill: storage consumption explosion

- Since no alternatives available, wrote something new: *static-container-registry*
- Given set of ‘exported’ container images, deduplicate layers (*hardlink*), then run script to generate *nginx* config include snippet, and serve static files as-is, act as ‘good enough’ registry API
- Tested with Docker, *containerd*, *cri-o*, *skopeo*

https://github.com/NicolasT/static-container-registry/

MetalK8s 2.x: Changed Architecture

- Everything centered around Kubernetes
  - Use K8s *Nodes* as ‘inventory’
  - Use K8s *Nodes* as source-of-truth for server roles
  - Deploy cluster services (Yum repo, registry,...) as K8s/*kubelet*-managed Pods: central view of cluster, unified log management, metrics gathering,...
  - Boils down to *etcd* cluster availability: backup and disaster-recovery tooling available
- Cluster management API: Kubernetes API + SaltAPI
  - Integrated SaltAPI authn with K8s authn, i.e. can do OIDC
  - Integrated SaltAPI authz with K8s RBAC, i.e. unified experience
  - Allows SaltAPI access to MetalK8s services requiring this through *ServiceAccounts*
- Embrace K8s model: declare first in model, then realize: nodes, storage, applications,...

MetalK8s 2.x: Covering Enterprise Datacenter Needs

- 100% offline installation
  - System packages
  - Containers
- Networking
  - Discern control-plane and workload-plane networks
  - Control-plane HA either *keepalived* managed by MetalK8s, or external VIP/LB
- RHEL7 / CentOS7
  - Ubuntu 18.04 LTS in progress for community purposes
  - RHEL8 / CentOS8 support planned
- Integrate with enterprise authn/authz
- Custom cluster-management UI
- Focus on ‘real hardware’, not cloud VMs
  - Integrated SMART disk monitoring, iLO server monitoring,...

SCALITY METALK8S PLATFORM

EN admin

Monitoring

Nodes

Nodes

+ Create a New Node

Name	Status	Deployment	Roles	MetalK8s Version
bootstrap	Ready		Bootstrap	2.4.0-dev

SCALITY METALK8S PLATFORM

EN admin

Monitoring

Nodes

Nodes > Create a New Node

New Node Data

Name

MetalK8s Version 2.4.0-dev

Roles

- [x] Workload Plane
- [ ] Control Plane
- [ ] Infra

New Node Access

SSH User

Hostname or IP

SSH Port 22

SSH Key Path /etc/metalk8s/pki/salt-bootstra

Sudo Required

Cancel Create

MetalK8s 2.x: Storage Provisioning

- Still focus on local disk manipulation
- Pure LVM2 not sufficient due to application needs
- Embrace K8s model + Salt:
  - Create *Volume* custom resource objects in K8s
  - Controller / ‘operator’ acts on *Volume* CRUDs
  - Every *Volume* has a *StorageClass* which defines FS, *mkfs* options, *mount* options,...
  - Calls into SaltAPI to realize ‘physical’ volumes, or LVM LVs, or …
    - Currently: *rawBlockDevice* and *sparseLoopDevice*
    - Coming: configure RAID controllers, based on existing Scality Salt tooling, incl. RAID controller management
  - Once ‘physical’ volume created and formatted, create K8s *PersistentVolume*, expose to workloads
- Not using CSI

SCALITY METALK8S PLATFORM

EN admin

Monitoring

Nodes

Nodes > bootstrap

Details Volumes Pods

Search...

Name	Status	Bound	Storage Capacity	Storage Class	Creation Time	Action
bootstrap-alertmanager	Available	Yes	1Gi	metalk8s-prometheus	9/23/2019 11:26:00 AM	
bootstrap-prometheus	Available	Yes	10Gi	metalk8s-prometheus	9/23/2019 11:26:00 AM	

SCALITY METALK8S PLATFORM

EN admin

Monitoring

Nodes

Nodes > bootstrap > bootstrap-prometheus

Detailed Information

Name	bootstrap-prometheus
Status	Available
Size	10Gi
Type	sparseLoopDevice
Bound	Yes
Storage Class	metalk8s-prometheus
Path	Not applicable
Access Mode	ReadWriteOnce
Mount Options	rw, discard
Creation Time	9/23/2019 11:26:00 AM

nicolas@beta:~/Projects/metalk8s

```
[root@bootstrap ~]# kubectl get volumes
NAME                    NODE        STORAGECLASS
bootstrap-alertmanager  bootstrap   metalk8s-prometheus
bootstrap-prometheus    bootstrap   metalk8s-prometheus
[root@bootstrap ~]# kubectl get volume bootstrap-prometheus -o yaml
apiVersion: storage.metalk8s.scality.com/v1alpha1
kind: Volume
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"storage.metalk8s.scality.com/v1alpha1","kind":"Volume","metadata":{"annotations":{},"name":"bootstrap-prometheus"},"spec":{"nodeName":"bootstrap","spars
eLoopDevice":{"size":"10Gi"},"storageClassName":"metalk8s-prometheus","template":{"metadata":{"labels":{"app.kubernetes.io/name":"prometheus-operator-prometheus"}}}}}
  creationTimestamp: "2019-09-23T18:26:00Z"
  finalizers:
  - storage.metalk8s.scality.com/volume-protection
  generation: 2
  name: bootstrap-prometheus
  resourceVersion: "1216"
  selfLink: /apis/storage.metalk8s.scality.com/v1alpha1/volumes/bootstrap-prometheus
  uid: c1e71f60-9d3a-4ca5-bdc4-1812eb29721b
spec:
  nodeName: bootstrap
  sparseLoopDevice:
    size: 10Gi
  storageClassName: metalk8s-prometheus
  template:
    metadata:
      creationTimestamp: null
      labels:
        app.kubernetes.io/name: prometheus-operator-prometheus
    spec: {}
status:
  phase: Available
[root@bootstrap ~]#
```

0 1*Z ssh ↑ 6h 21m 23s 2.0 2.2 2.2 2019-09-23 23:32 beta.nicolast.be

nicolas@beta:~/Projects/metalk8s

```
[root@bootstrap ~]# kubectl describe pv bootstrap-prometheus
Name:              bootstrap-prometheus
Labels:            app.kubernetes.io/name=prometheus-operator-prometheus
Annotations:       pv.kubernetes.io/bound-by-controller: yes
Finalizers:        [storage.metalk8s.scality.com/volume-protection kubernetes.io/pv-protection]
StorageClass:      metalk8s-prometheus
Status:            Bound
Claim:             metalk8s-monitoring/prometheus-prometheus-operator-prometheus-db-prometheus-prometheus-operator-prometheus-0
Reclaim Policy:    Retain
Access Modes:      RWO
VolumeMode:        Filesystem
Capacity:          10Gi
Node Affinity:
  Required Terms:
    Term 0:        kubernetes.io/hostname in [bootstrap]
Message:
Source:
    Type:  LocalVolume (a persistent volume backed by local storage on a node)
    Path:  /dev/disk/by-uuid/c1e71f60-9d3a-4ca5-bdc4-1812eb29721b
Events:    <none>
[root@bootstrap ~]#
```

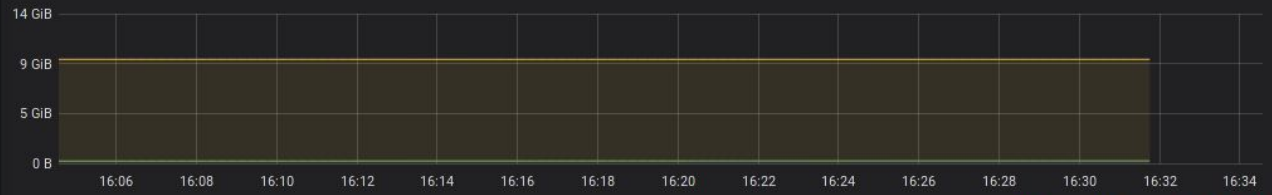
0 1*Z ssh ↑ 6h 27m 26s 2.1 2.1 2.1 2019-09-23 23:38 beta.nicolast.be

Kubernetes / Persistent Volumes

Last 30 minutes

datasource: Prometheus | Namespace: metalk8s-monitoring | PersistentVolumeClaim: prometheus-prometheus-operator-prometheus-db-prometheus-prometheus-operator-prometheus-0

Volume Space Usage



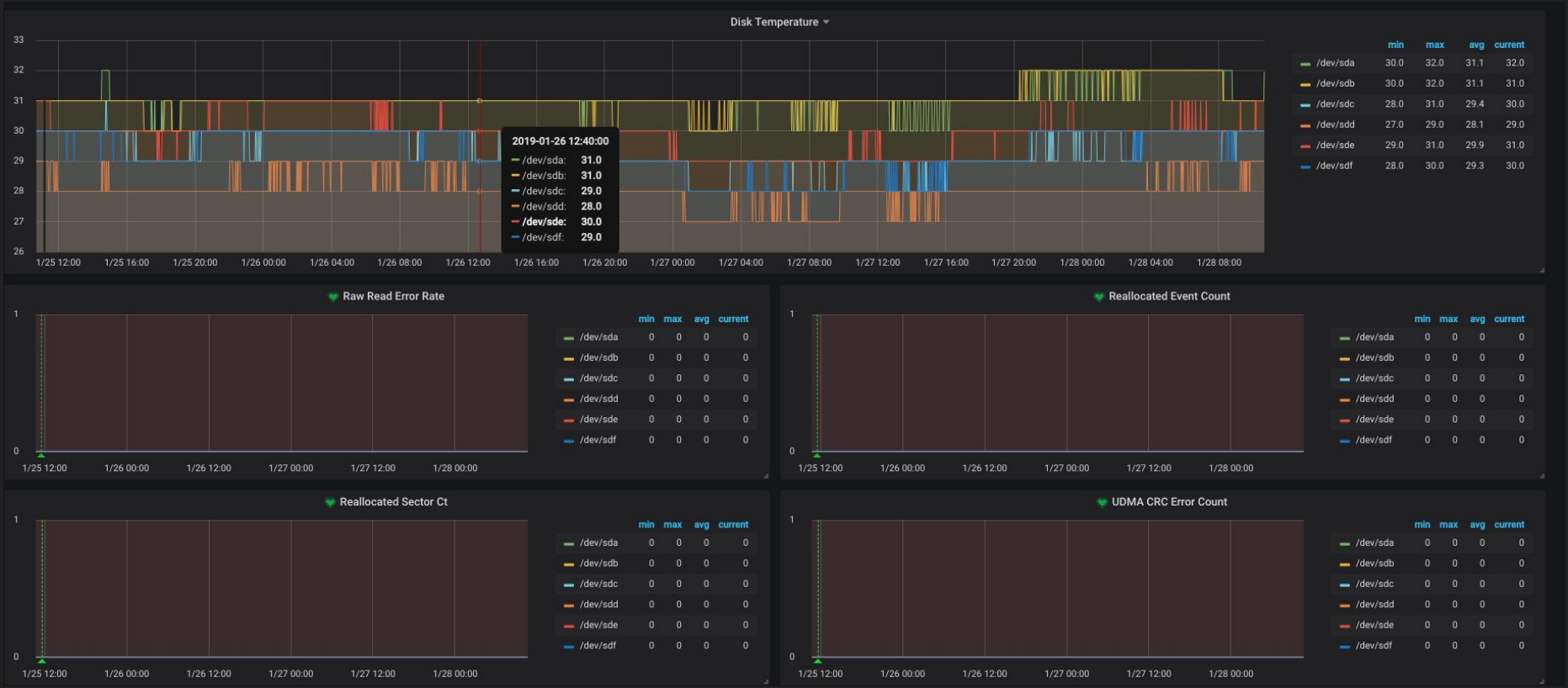
	min	max	avg	current
Used Space	243 MiB	262 MiB	254 MiB	
Free Space	9.46 GiB	9.48 GiB	9.47 GiB	

Volume Space Usage

Volume inodes Usage

	min	max	avg	current
Used inodes	30	30	30	
Free inodes	655330	655330	655330	

Volume inodes Usage


Disk Temperature
33
32
31
30
29
28
27
26
1/25 12:00
1/25 16:00
1/25 20:00
1/26 00:00
1/26 04:00
1/26 08:00
1/26 12:00
1/26 16:00
1/26 20:00
1/27 00:00
1/27 04:00
1/27 08:00
1/27 12:00
1/27 16:00
1/27 20:00
1/28 00:00
1/28 04:00
1/28 08:00
2019-01-26 12:40:00
/dev/sda: 31.0
/dev/sdb: 31.0
/dev/sdc: 29.0
/dev/sdd: 28.0
/dev/sde: 30.0
/dev/sdf: 29.0
min max avg current
/dev/sda 30.0 32.0 31.1 32.0
/dev/sdb 30.0 32.0 31.1 31.0
/dev/sdc 28.0 31.0 29.4 30.0
/dev/sdd 27.0 29.0 28.1 29.0
/dev/sde 29.0 31.0 29.9 31.0
/dev/sdf 28.0 30.0 29.3 30.0
Raw Read Error Rate
1
0
1/25 12:00
1/26 00:00
1/26 12:00
1/27 00:00
1/27 12:00
1/28 00:00
min max avg current
/dev/sda 0 0 0 0
/dev/sdb 0 0 0 0
/dev/sdc 0 0 0 0
/dev/sdd 0 0 0 0
/dev/sde 0 0 0 0
/dev/sdf 0 0 0 0
Reallocated Event Count
1
0
1/25 12:00
1/26 00:00
1/26 12:00
1/27 00:00
1/27 12:00
1/28 00:00
min max avg current
/dev/sda 0 0 0 0
/dev/sdb 0 0 0 0
/dev/sdc 0 0 0 0
/dev/sdd 0 0 0 0
/dev/sde 0 0 0 0
/dev/sdf 0 0 0 0
Reallocated Sector Ct
1
0
1/25 12:00
1/26 00:00
1/26 12:00
1/27 00:00
1/27 12:00
1/28 00:00
min max avg current
/dev/sda 0 0 0 0
/dev/sdb 0 0 0 0
/dev/sdc 0 0 0 0
/dev/sdd 0 0 0 0
/dev/sde 0 0 0 0
/dev/sdf 0 0 0 0
UDMA CRC Error Count
1
0
1/25 12:00
1/26 00:00
1/26 12:00
1/27 00:00
1/27 12:00
1/28 00:00
min max avg current
/dev/sda 0 0 0 0
/dev/sdb 0 0 0 0
/dev/sdc 0 0 0 0
/dev/sdd 0 0 0 0
/dev/sde 0 0 0 0
/dev/sdf 0 0 0 0

SCALITY

SCALITY METALK8S PLATFORM

EN admin

Monitoring

Nodes

Nodes > bootstrap > Create a New Volume

Name

Storage Class metalk8s-prometheus

Type SparseLoopDevice

Volume Capacity GiB

Cancel Create

MetalK8s 2.x: Deploying Solutions

- K8s cluster not ‘end goal’, how to deploy actual product(s)?
- Main concept: operators
  - A solution (e.g. Zenko) brings an *operator* that can deploy, manage, lifecycle solution instance(s)
- Solutions ship as ISO images containing container images and metadata
  - Somewhat similar to CNAB ‘thick’ bundles
- ISOs ‘imported’ in cluster (on *bootstrap* node)
  - Expose containers in registry
  - Deploy operator & custom UI
  - Create solution-specific *StorageClass*es
- Think of this as the *application store* of your cluster

MetalK8s 2.x: Deploying Solutions

- Solution instances created/deployed, upgraded, downgraded,... by user, automated by operator
  - Through K8s (extended) API, CLI, solution-specific UI
- Metrics captured by cluster-provided Prometheus, monitored by cluster-provided AlertManager & Grafana
  - Solution deployment includes custom dashboards and alerting rules

MetalK8s 2.x: Quickstart

- Install Vagrant and VirtualBox
- `git clone --branch development/2.4 \`
  `https://github.com/scality/metalk8s.git`
- `./doit.sh vagrant_up`

MetalK8s 2.x: The road forward

- Integrate log aggregation
- Increase documentation coverage
- Extend UI
- Storage:
  - Device discovery
  - Integrate existing RAID-controller-fronted-disks automation in MetalK8s
  - Considering using SNIA Swordfish for discovery and provisioning
  - Support non-traditional device access (SPDK, CNS,...)
- Extended host/device/network monitoring
- Other CNIs (sriov, DPDK), Istio service-mesh, Jaeger tracing, OpenPolicyAgent,...
- Experimenting with built-in node netboot: PXE, boot-from-RAMdisk/livenet

SCALITY METALK8S

AN OPINIONATED KUBERNETES DISTRIBUTION WITH A FOCUS ON LONG-TERM ON-PREM DEPLOYMENTS

https://github.com/scality/metalk8s

Nicolas Trangez - Senior Architect
nicolas.trangez@scality.com
@eikke | @scality | @zenko

SCALITY