Container Orchestration with Kubernetes on SUSE® Linux

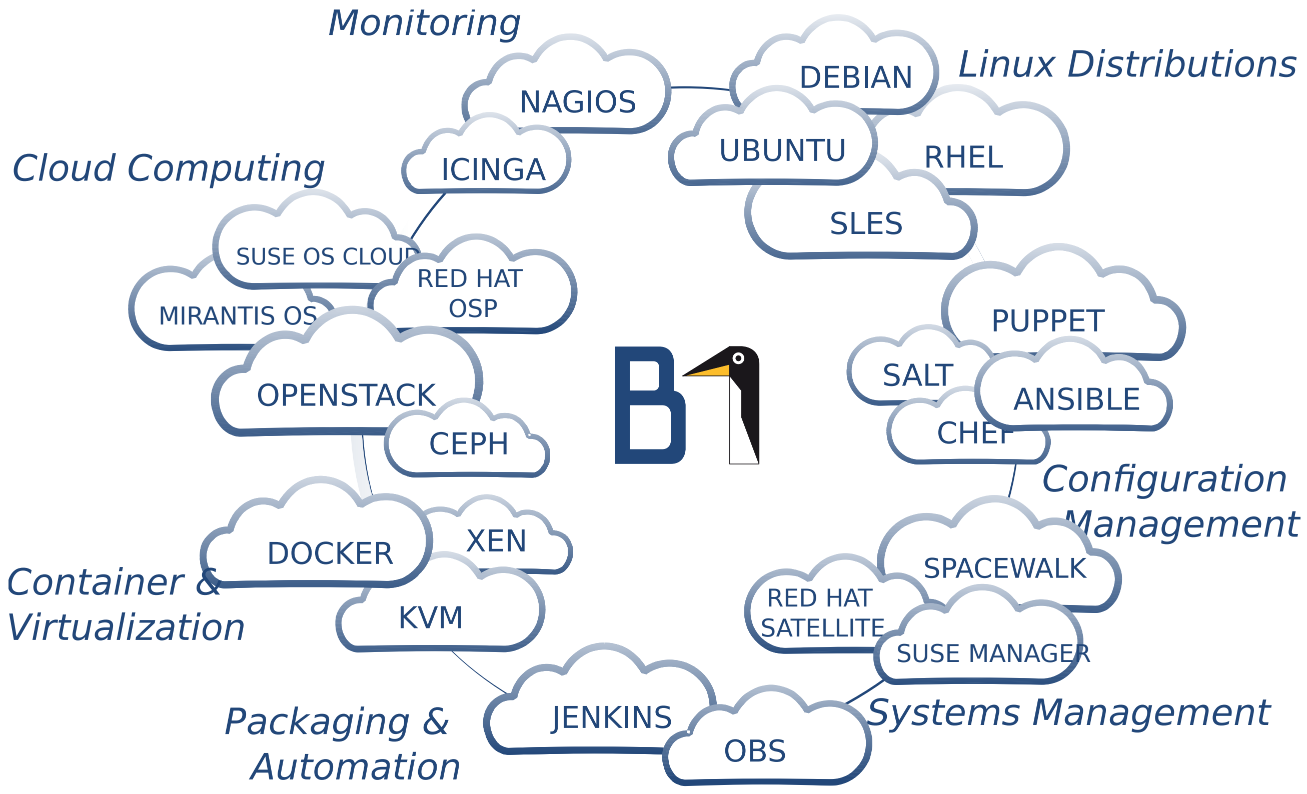
André Steincke
Linux Consultant & Trainer
B1 Systems GmbH

Michael Steinfurth
Linux / Unix Consultant & Trainer
B1 Systems GmbH

Introducing B1 Systems

- founded in 2004
- operating both nationally & internationally
- about 100 employees
- vendor-independent (hardware & software)
- focus:
  - consulting
  - support
  - development
  - training
  - operations
  - solutions
- offices in Rockolding, Berlin, Cologne & Dresden

Areas of expertise



The on-premises setup

Overview

- etcd-cluster
- overlay-network (e.g. *flannel*)
- open source Docker container engine
- Kubernetes master and worker nodes
- shared storage (NFS)
- image registry

Etcd cluster

- necessary for Kubernetes
- also with *flanneld*
- high availability is necessary (# nodes 3,5,7...)
- quorum majority needed
  - ➔ run on odd number of DCs (min. 3)
- part of the coreos project

Overlay network

- containers need to communicate on multiple nodes
- multiple solutions
  - *flannel*, weave plugin, opencontrail
- *flannel* integrates most easily + quite good performance
- part of the coreos project

Installation & Configuration

Packages and where to get them

- *etcd*, *flanneld*
- Kubernetes master
- Kubernetes worker
- Kubernetes client
- *docker*

- Open Build Service
- Kubic media DVD

Add repo & install packages

- Add repos

```
# mkdir /mnt/kubic
# mount openSUSE-Kubic-DVD-x86_64...iso \
   /mnt/kubic/
# zypper addrepo __PATHTOKUBIC__ kubic1
# zypper ref
```

- Install packages

```
# zypper install etcd etcdctl flannel \
   kubernetes-master kubernetes-node kubernetes-client
```

etcd configuration

- Do an one time etcd cluster initialization and edit config permanently
- Do on each node (adapt ip addresses):

```
# bash etcd.sh (initial start) wait 1 min → stop script
# chown -R etcd: /var/lib/etcd
# vi /etc/sysconfig/etcd
```

```
ETCD_NAME="default"
ETCD_DATA_DIR="/var/lib/etcd/default.etcd"
ETCD_LISTEN_PEER_URLS="http://10.1.1.101:2380"
ETCD_LISTEN_CLIENT_URLS="http://localhost:2379,http:
//10.1.1.101:2379"
ETCD_ADVERTISE_CLIENT_URLS="http://10.1.1.101:2379"
```

etcd cluster start

- Start etcd on each node and check cluster health state

```
# systemctl start etcd
# etcdctl cluster-health
```

Flannel configuration

- Set *flannel* config on each node (can ssh copy config)
- edit: FLANNEL_ETCD_ENDPOINTS FLANNEL_ETCD_KEY

```
# etcdctl set /network/config \
  '{ "Network": "172.19.0.0/16", "Backend": { "type": "host-gw"} }'
# vi /etc/sysconfig/flannel
```

```
# Flanneld configuration options

FLANNEL_ETCD_ENDPOINTS="http://k8snode1:2379,http://
k8snode2:2379,http://k8snode3:2379"
FLANNEL_ETCD_KEY="/network"
FLANNEL_OPTIONS="-iface eth0"
```

flanneld start

- Start *flanneld*

```
# systemctl start flanneld
# systemctl is-active flanneld
```

Kubernetes configuration master (1/2)

- Apiserver
  - Remove service account
  - Add *etcd* cluster
  - Nodeport range for external access towards applications
  - Set service-cluster-ip-range

Configure the apiserver

- Edit the file /etc/kubernetes/apiserver
  - adapt bind-address for each node

```
KUBE_API_ADDRESS="--insecure-bind-address=0.0.0.0"
KUBE_API_PORT="--insecure-port=8080"
KUBE_ETCD_SERVERS="--etcd-
servers=http://k8snode1:2379, \
http://k8snode2:2379,http://k8snode3:2379"

KUBE_SERVICE_ADDRESSES="--service-cluster-ip-
range=172.18.0.0/16"

KUBE_ADMISSION_CONTROL="—-admission-
control=NamespaceLifecycle,LimitRanger,DefaultStorag
eClass,ResourceQuota"

KUBE_API_ARGS="-—service-node-port-range=30000-40000
--bind-address=10.1.1.101"
```

Kubernetes configuration master (2/2)

- Scheduler
  - no change needed
- Controller-manager
  - Remove service account feature
  - Edit the file /etc/kubernetes/controller-manager

```
KUBE_CONTROLLER_MANAGER_ARGS="--cluster-name=mykubecluster"
```

Kubernetes configuration worker

- Edit the file /etc/kubernetes/kubelet

```
KUBELET_ADDRESS="--address=0.0.0.0"
# KUBELET_HOSTNAME="--hostname-override=127.0.0.1"
```

- Edit the file /etc/sysconfig/docker

```
DOCKER_OPTS="--ip-masq=false --iptables=false"
```

Kubernetes shared configuration

/etc/kubernetes/config

- all components share this configuration
- each individual component’s config file settings has higher priority

```
KUBE_LOGTOSTDERR="--logtostderr=true"
KUBE_LOG_LEVEL="--v=0"
KUBE_ALLOW_PRIV="--allow-privileged=false"
KUBE_MASTER="--master=http://127.0.0.1:8080"
```

Kubernetes start & check

```
# systemctl start docker
# systemctl start kube-apiserver.service \
  kube-scheduler kube-controller-manager
  kube-proxy kubelet
# systemctl is-active "kube*" "docker"
```

Exercise 1

Exercise 1

1) Change the port range.

2) Change the service ip address range.

3) OPTIONAL: Change the timeout for restarting pods of dead nodes.

The Kubernetes console client – Quick intro into *kubectl*

Kubectl (1/2)

- List nodes
- get detailed information about nodes in cluster

```
# kubectl get nodes
# kubectl describe nodes
```

- Make node unschedulable (+ draining pods) and scheduable

```
# kubectl cordon nodes
# kubectl drain nodes
# kubectl uncordon nodes
```

Kubectl (2/2)

- Manage entities and applications

```
# kubectl get (pods|replicasets|service) (-o wide|yaml) (-w)
# kubectl create -f __ENTITY.YAML__
# kubectl delete (pod|replicaset|service) __ENTITYNAME__
```

- Use for debugging of platform and applications

```
# kubectl get events
# kubectl describe node (__NODE1__)
# kubectl logs (-f) __POD_NAME__
```

Applications with Kubernetes

Pods

- smallest entity
- one IP address
- consists out of minimum two containers
  - 1 infra container (keeps network namespace)
  - min. 1 app container (e.g. *nginx*)
- other entities can control pods by matching labels
- Temporary state
  - Mentality:
    1) Delete
    2) Restart
    3) works the same as before (different parameters: ip address, hostname)

Pod YAML (Example)

- create a file `/home/tux/k8s/pod.yaml` with the following content

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: nginx
  name: mynginx
  namespace: default
spec:
  containers:
  - name: nginxcont
    image: nginx:latest
    imagePullPolicy: Always
    env:
    - name: LOGDIR
      value: /srv/logs
```

Start a pod and get information about the pods

- To start a pod use the following command

```
# kubectl create -f /home/tux/k8s/pod.yaml
```

- Get information about pods

```
# kubectl get pods
# kubectl get pods -o wide
# kubectl get pods mynginx -o yaml
```

Replicasets / ReplicationController

- controlling entity
- keeps track of running (number) pods
- find pods to control by label

```
run = nginx
stage = test
```

- Replicaset same as ReplicationController
- Replicaset additonally supports match label out of given quantities

```
stage in (dev, test)
```

- keyword: `replicas` defines number of running pods

Deployments

- Special entity to control versioning of other entities (e.g. replicasets) and your applications
- Based on used images
- Updates, downgrades
- Supports different strategies:
  - *Rolling update* ,"step by step" e.g.:
    1) "*stop one oldversioned pod*" → "*start one newversioned pod*"
    2) "*stop next oldversioned pod*" → "*start ...*"
  - *Recreate*
    1) *"stop all oldversioned pods"*
    2) *"start all newversioned pods"*
- undo / redo operations supported

Start a deployment and get information about it

- To create the deployment use the following command

```
# kubectl create -f /home/tux/k8s/depl.yaml
```

- Get information about pods

```
# kubectl get depl -o yaml
# kubectl get rs
# kubectl get pods
```

Deployment YAML (Example)

```yaml
apiVersion: v1
kind: Deployment
metadata:
  name: testmicroapplication-deployment
spec:
  replicas: 1
  strategy:
    type: RollingUpdate
  template:
    spec:
      restartPolicy: Always
      containers:
      - name: testmicroapplication
      image: registry/apps/testmicroapplication:B123
      env:
```

Advanced deployment watching

1) Start on one terminal a watch on changing pods

```
# kubectl get pods -o wide -w
```

2) Open another terminal and delete a running pod from the depl.

- Choose a pod from the specific deployment (label)

```
# kubectl get pod -l run=mynginxdeployment
# kubectl delete __PODNAME__
```

3) Watch the output of the first terminal

Externally accessibility for applications

Services

- Entities representing one application to other applications
- also the way to present your application to outside world
- Uses combination of virtual IP addresses + port + transportprotocol
- doesn’t change on upgrade or downgrade of your apps
- Hides internal addresses of applications
- Supports loadbalancing to several pods of same kind (same app)

Start a service and get information about it

- To create the deployment use the following command

```
# kubectl create -f /home/tux/k8s/svc.yaml
```

- Get information about pods

```
# curl -vvv http://localhost:32222
# kubectl get svc -o yaml
# kubectl get ep
```

External service YAML

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  namespace: default
spec:
  selector:
    run: nginxpod
  ports:
  - name: nginx-port
    port: 8080
    targetPort: 80
    protocol: TCP
    nodePort: 32222
  type: NodePort
```

Session stickyness service YAML (example)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
  namespace: default
spec:
  selector:
    run: nginxpod
  ports:
  - name: nginx-port
    port: 8080
    targetPort: 80
    protocol: TCP
    nodePort: 33333
  type: NodePort
  sessionAffinity: ClientIP
```

Exercise 2

Exercise 2

1) Create a new deployment of 3 running apache webservers.

2) Create a service for this deployment accessible from external networks.

3) OPTIONAL: Make the application access sticky for client-IP-Address.

Provide variable data to your applications

Configmaps

- get variable data into your pod
- cluster wide
- independent from other entities
- different was of config data creation
- representing data in different ways in pods
  - filebased
  - environment variables (since 1.6)

Create configmap from file + usage in pod

```
# kubectl create configmap dbconfig \
  -–from-file=/home/tux/k8s/dbconfig.properties
```

```
spec:
  containers:
  - name: nginxcont
    [...]
    volumeMounts:
      - name: config-volume
        mountPath: /etc/config
  volumes:
    - name: config-volume
      configMap:
        name: dbconfig
```

configmap for ENV variables in pod

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dbconfig-env
  namespace: default
data:
  SERVER: db.example.org
  USERNAME: dbuser
  DATABASE: test
```

```
  Containers:
  [...]
    envFrom:
    - configMapRef:
       name: dbconfig-env
```

Secrets

- centralized way to save sensitive data
- passwords, tokens, keys
- not in cleartext, but base64
- Better way of safing secure information, than in a pod
- Similar to configmaps
  - key/value from files or standard input
- Data presented as
  - file (name → key, content → value)
  - ENV variable (name → userdef., content → value)

Create secret from file + vol usage in pod

```
# kubectl create secret generic secret1 \
  -–from-file=/home/tux/k8s/password
```

```
spec:
  containers:
  - name: nginxcont
    [...]
      volumeMounts:
      - name: secretvol
        mountPath: /etc/config/access
  volumes:
    - name: secretvol
      secret:
        secretName: secret1
```

secret as ENV variables inside pod

```
Containers:
[…]
  env:
   - name: PASSWORD
     valueFrom:
       secretKeyRef:
         name: secret1
         key: password
```

Volumes

- way to save data outside the r/w-layer
- persistent
  - Shared filesystem needed to save cluster wide
  - Usage of external storage provider (cephfs, netapp)
  - hostpath option, hostbased mount, every host the same

- temporary
  - lifetime = container lifetime
  - emptydir
  - think of a typical linux like “tmp” directory

More volumes

- secrets
  - used to save secret data / passwords
- configmaps
  - used to save configuration data cluster wide

Exercise 3

Exercise 3

1) Extend your deployment by a self provided configuration (configmap Volume) on your webservers.

2) Scale your existing deployment up to 5 replicas.

3) OPTIONAL: Protect the website access with basic auth by using the kubernetes secrets function connected to your deployment.

Additional Entities

Daemonsets

- Used to make a pod run on each kubernetes node
- Examples are
  - logging services, reading logs from nodes
  - Kubernetes components themself (apiserver,scheduler,flannel)

daemonset YAML (Example 1/2)

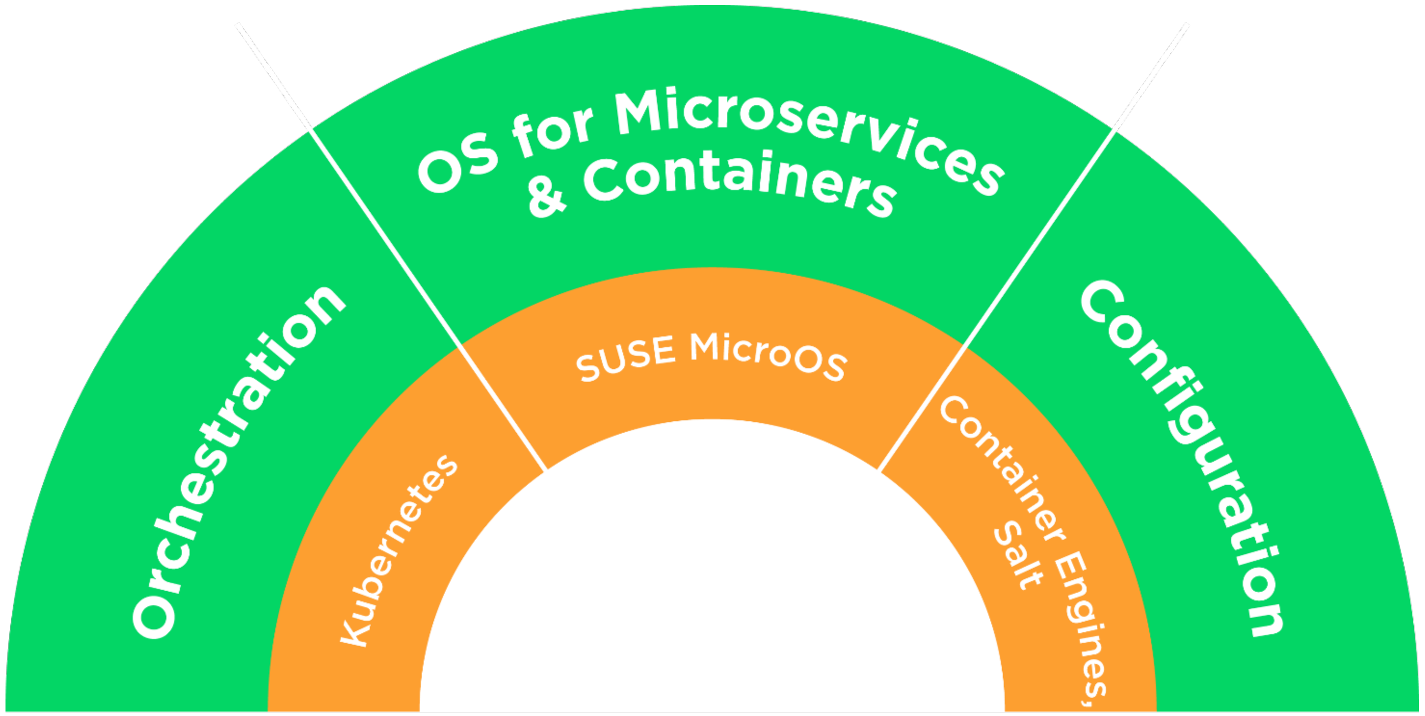
```
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: fluentd-daemonset
  namespace: kube-system
spec:
  template:
    metadata:
      labels:
        run: fluentd
        env: tst
      name: fluentd
    spec:
    ...
```

daemonset YAML (Example 2/2)

```
...
    spec:
      restartPolicy: Always
      containers:
      - name: fluentd
        image: gcr.io/google_containers/fluentd-
elasticsearch:1.19
        env:
        - name: FLUENTD_ARGS
          value: -qq
```

Similarities and Differences to SUSE CaaS Platform

3 Key Technology Components



SUSE CaaS Platform

What is SUSE MicroOS

A purpose built Operating System designed for **microservices** & **containers** and optimized for **large deployments**.

Term “Micro” in MicroOS signifies Microservices.

Key Features

An always up-to-date Operating System

An easy to manage/upgrade OS

Easily setup/manage a cluster of nodes

Scalable — up to 1000s of nodes

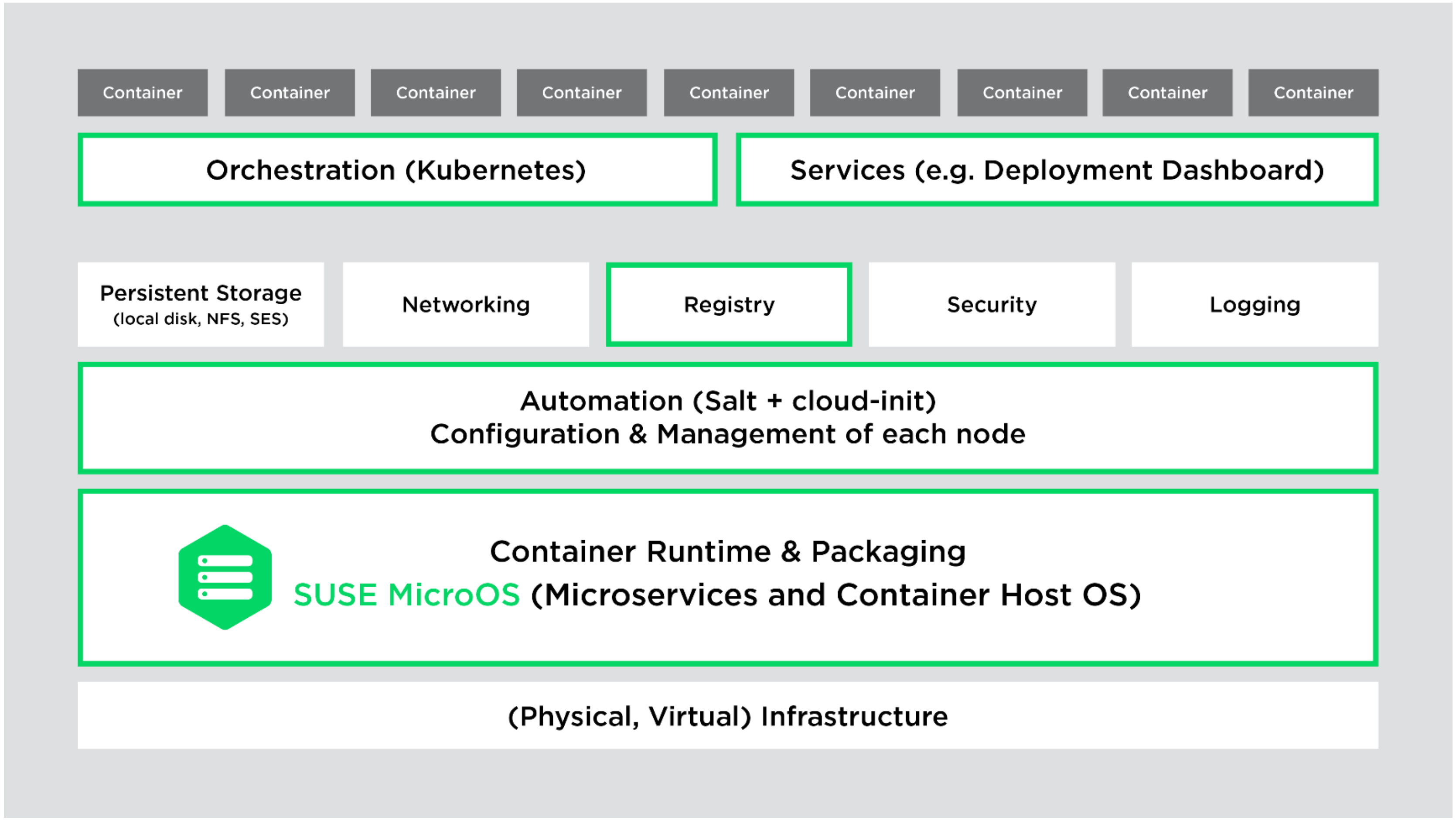
What SUSE MicroOS is NOT

SUSE MicroOS is **not** a separate product.
It is only available as part of SUSE CaaS Platform.

SUSE MicroOS is **not** related to JeOS.

SUSE MicroOS is **not** a compressed version of SUSE Linux Enterprise Server.
It is created from scratch using SUSE Linux Enterprise Server components. That's why it inherits the enterprise grade quality, security certifications of the modules and technology such as btrfs.

SUSE CaaS Platform – Stack View



Container
Container
Container
Container
Container
Container
Container
Container
Container
Orchestration (Kubernetes)
Services (e.g. Deployment Dashboard)
Persistent Storage (local disk, NFS, SES)
Networking
Registry
Security
Logging
Automation (Salt + cloud-init)
Configuration & Management of each node
Container Runtime & Packaging
SUSE MicroOS (Microservices and Container Host OS)
(Physical, Virtual) Infrastructure

Thank you!

Reference links:
https://kubernetes.io/docs/api-reference/v1.6/

Logos: Coreos Project (Apache License)

SUSE®

We adapt. You succeed.

Fallback on broken quick setup

- In case setup brakes
- Use prepared patch to configure all systems
- Clean etcd, kubernetes

```
# tar -xzf kubernetes.tar -C /etc
# patch -p0 -d /etc/ -i __SECRET_PATH__
```