Technische Hochschule Rosenheim
Technical University of Applied Sciences

Microservices

Container - Kubernetes (K8s)

Content

- Challenges in distributed container platforms
- Container orchestration platforms
- Kubernetes

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Container orchestration platforms

Why clustering & orchestration platforms?

- High availability
- Scalability
- "Scale out" instead of "scale up"
- Simplify the maintenance of cluster nodes (updating, replacement, hardware maintenance,...)

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Challenges in distributed container platforms

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Persistence

Persistence - challenges

How to store the data?

- NFS
- vSphere volumes
- SAN storages (EMC, HPE 3PAR,…)
- Cloud storages…

Persistence - tasks

Choose one or multiple persistence drivers that are available for Docker and K8s:

- Azure File and Disk
- AWS EBS
- GCE PD
- Glusterfs
- Ceph RBD
- vSphere

Persistence - conclusion

- The choice of a persistence driver depends on the environment (on premise vs. cloud)
- Which level of latency is required (SSD vs. HDD storage)
- Which kind of access is required? (Exclusiv vs. shared access of volumes)
- Is there already storage infrastructure available?
- What kind of deployment is planned? (Development, staging, production)

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Networking

Networking - challenges

- Load balancing
- Service publishing (especially from outside the cluster)
- Service discovery (cross service calls)

Networking - tasks

- Application publishing - how to route traffic to container running within a cluster system
- Cross container communication - how to access a container running on a different node transparently
- How to isolate applications within the cluster system - how to run multiple environments (e.g. staging and production) side-by-side
- Service Discovery - how to access other services without knowing their IP addresses (and probably the port used by the container)

Configuration and secrets

Configuration and secrets - tasks

- Central management of configuration elements
- Mounting configuration elements into container instances (via file system mounts or environment variables)
- Sharing of configuration elements across nodes and container instances
- Storing secret variables e.g. AWS credentials, certificates,...

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Scaling

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Scaling - challenges

- When to scale up and down?
- When to consider the new instance for new requests?
- How to avoid service interruptions by scaling down while an instance is still processing a request?
- How to automatically reconfigure load balancers?

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Hardware access

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Hardware access - challenges

- How to enable containers to access hardware devices e.g. GPUs
- How to ensure that container is scheduled on a node where the expected device is available
- How to manage drivers so that the driver used within the container equals the one of the host system (if not provided by the kernel)

Hardware access - solutions

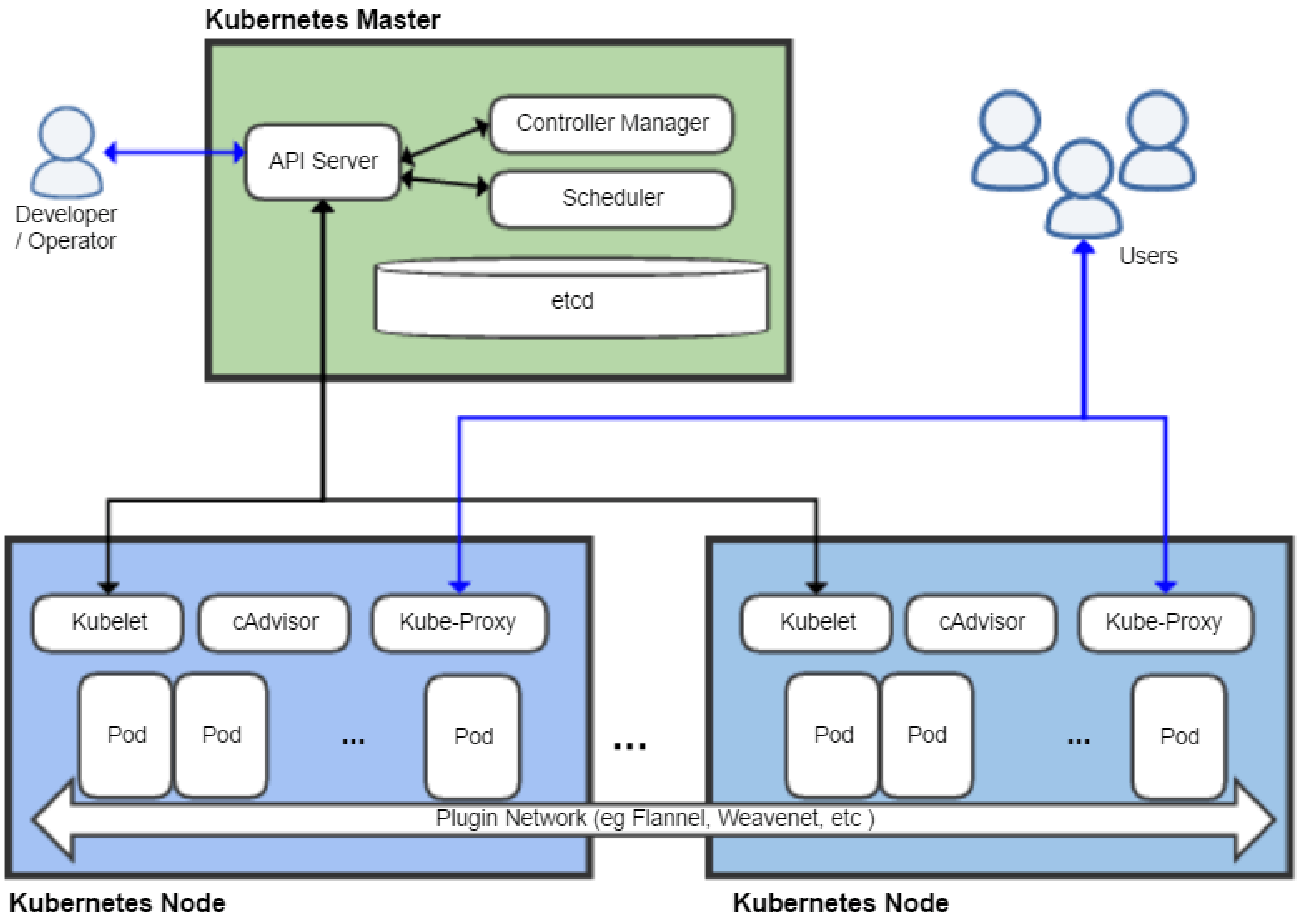
- Device plugins API to abstract hardware access of containers
- Device plugins for GPUs:
  - NVIDIA/k8s-device-plugin
  - intel/intel-device-plugins-for-kubernetes
  - RadeonOpenCompute/k8s-device-plugin
- Some of them (like the Nvidia GPU plugin) are also available for Docker (nvidia-docker) to enable local debugging

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Kubernetes

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Kubernetes architecture


Kubernetes Master
Developer / Operator
API Server
Controller Manager
Scheduler
etcd
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
...
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node

Kubernetes basic elements

- Pods
- Deployments
- Services
- Ingresses
- Jobs
- ConfigMaps
- Secrets
- HPA
- PDB
- ...

Kubernetes basic elements - Pods

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
```

Kubernetes basic elements - Pods

- Pods are the most atomic element in Kubernetes
- A Pod might contain multiple containers
- A Pod is always scheduled on a single worker/minion
- Containers within a Pod can access each other by calling `localhost` or `127.0.0.1`
- Because all containers of a Pod are scheduled on a single node local volume mounts are available for all containers

Technische Hochschule Rosenheim
Technical University of Applied Sciences

Kubernetes basic elements - Deployments

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: MyApp
spec:
  replicas: 3
  selector:
    matchLabels:
      app: MyApp
  template:
    metadata:
      labels:
        app: MyApp
```

Kubernetes basic elements - Deployments

- Deployments are a high level approach to deploy containerized applications to a Kubernetes cluster
- A deployment triggers always the creation of a `ReplicationController`
- The corresponding `ReplicationController` takes care of the Pod creation for the Deployment
- Because a Deployment 'includes' a `ReplicationController` it is possible to set the number of `replicas` created when the Deployment is created (defaults to 1)
- The `spec` element of a Deployment contains the `template` used to create new Pods (including metadata,...)

Kubernetes basic elements - Services

```yaml
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80
```

Kubernetes basic elements - Services

- Every service has a DNS A record within a Kubernetes cluster - default is `<service-name>.<namespace-name>.svc.cluster.local` (and a SRV record but more on that later)
- Services are the Kubernetes native way of load balancing
- Services normally have a `selector` spec to determine which pods should be used to redirect the traffic to (based on the labels of a pod)
- Available types of a Service:
  - ClusterIP (default)
  - NodePort
  - LoadBalancer
  - ExternalName
- By using the type `ExternalName` it's possible to use the Kubernetes DNS to route traffic to external services

Kubernetes basic elements - Ingresses

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /testpath
        backend:
          serviceName: test
          servicePort: 80
```

Kubernetes basic elements - Ingresses

- An Ingress resource is basically just a routing rule for an Ingress Controller deployed to the cluster
- The Ingress controller is available e.g. by running a `NodePort` Service of Pods running an Ingress controller immplementation (e.g. kubernetes/ingress-nginx)
- Whenever a new Ingress resource is created the corresponding Ingress Controller should be reconfigured to be able to route traffic accordingly to the created Ingress resource
- A single Ingress resource might contain multiple rules (multiple hosts, multiple paths, multiple schemes,…)

Why Kubernetes for Microservices?

- Kubernetes already includes mechanisms for crucial tasks like service discovery and load balancing
- Kubernetes is able to dynamically scale deployments based on metrics like CPU and memory but also based on custom metrics e.g. avg. response time of HTTP requests
- Kubernetes itself can be scaled out to huge deployments of 500 nodes and thousands of containers
- There's a huge and still growing eco system of tools and frameworks available for Kubernetes (e.g. Knative, Spinnaker, Prometheus-Operator, Istio,...)