DSG-SOA-M 2024:
- Kubernetes -

Dr. Andreas Schönberger,
Johannes Manner

Distributed Systems Group
Faculty Information Systems and Applied Computer Science
University of Bamberg

What is Kubernetes?

- ❑ What is Kubernetes?
  “Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.”
  (https://kubernetes.io/)
- ❑ Why is K8s so developer friendly?
  “Containerization [and therefore K8s as well] transforms the data center of being machine-oriented to being application oriented.” [BGOBW2016, p.5]

“Kubernetes was built to radically change the way that applications are built and deployed in the cloud. Fundamentally, it was designed to give developers more velocity, efficiency, and agility.” [HBB2017, p.11]

Where does Kubernetes come from?

- ❑ All the systems are developed by Google Inc.

Borg*

- Built to manage long-running services and batch jobs
- Aim was to increase utilization
- Container support and isolation made sharing of resource possible
- Central component: Borg Master
- Remains the primary used container-management system within Goolge Inc. (?)

Omega*

- Offspring of Borg
- Consolidation of Borg's ecosystem
- Store state in a shared persistent store
- No central component: Decoupled the Borg master's functionality
- Established improvements are folded into Borg

Kubernetes

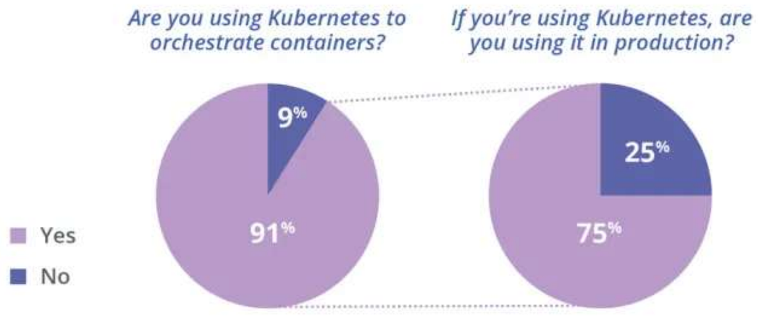
- Open source
- Designed by Google, maintained by the cloud native computing foundation (CNCF)
- Like Omega, has a shared persistent store
- Developed with a focus on external developers (public cloud provider)
- "its main design goal is to make it easy to deploy and manage complex distributed systems" [BGOBW2016 ,p.3]

https://static.googleusercontent.com/media/research.google.com/de//pubs/archive/44843.pdf

* Systems are closed source

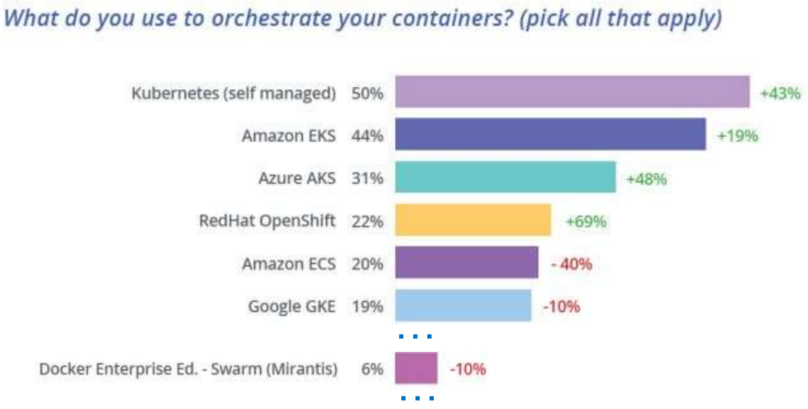
Some statistics . . .

- ❑ Survey of 400 IT and security professionals (2020)



K8s is **THE** tool to orchestrate containers

Managed K8s services become even more important compared to Elastic Container Service (ECS) or Docker Swarm



Figures from https://www.stackrox.com/kubernetes-adoption-security-and-market-share-for-containers/

The Success of K8s is Container Success

cf. Docker / Container lectures

- ❑ Utilization -> which means in the end $$$
- ❑ Containers provide isolation and dependency minimization -> only a few OS versions are needed
- ❑ Developers and operators do not worry about OS details
- ❑ Roll out of new hardware and OS is much easier since the container hides all details and the applications are not affected
- ❑ Self-healing of systems is much easier: If a container fails during execution or is unresponsive, the container management system spawns a new one and deletes the other
- ❑ Different pieces of a service can be developed independently from each other

→ and K8s gives the answer to professional operation of containers

Agenda – The Basic Concepts

cf. sources on literature slide

- ❑ Benefits of K8s
- ❑ Minikube
- ❑ K8s core components (K8s CC)
  - Master and Node
  - Deployment and App
  - Pods
  - Service
  - Labels and Annotations
  - Replica and DaemonSets
  - Data Integration and StatefulSets
- ❑ Outlook
- ❑ Case Study - CatService

What is the Benefit of K8s?

- ❑ Abstraction of infrastructure
  - ▪ Operation is much easier (abstraction of HW and OS details)
  - ▪ Easier portability due to higher level API (containers)
    (to achieve this avoid cloud managed services like DynamoDb etc.)
- ❑ Resource Efficiency
  - ▪ Cost of running a server (i.e. utilization, complexity)
  - ▪ Human resources to manage and setup infrastructure and systems
  - ▪ Containers are more lightweight than VMs or bare metal
- ❑ Development Speed
  - ▪ Iterative improvement – new services – zero downtime
  - ▪ Declarative configuration (what and not how) – self healing system (reliability)
- ❑ Supports scaling development
  - ▪ Decoupled architectures – independent scaling of parts of your system
  - ▪ Each team is responsible for a single microservice

[HBB2017, chapter 1]

Getting hands dirty - Minikube

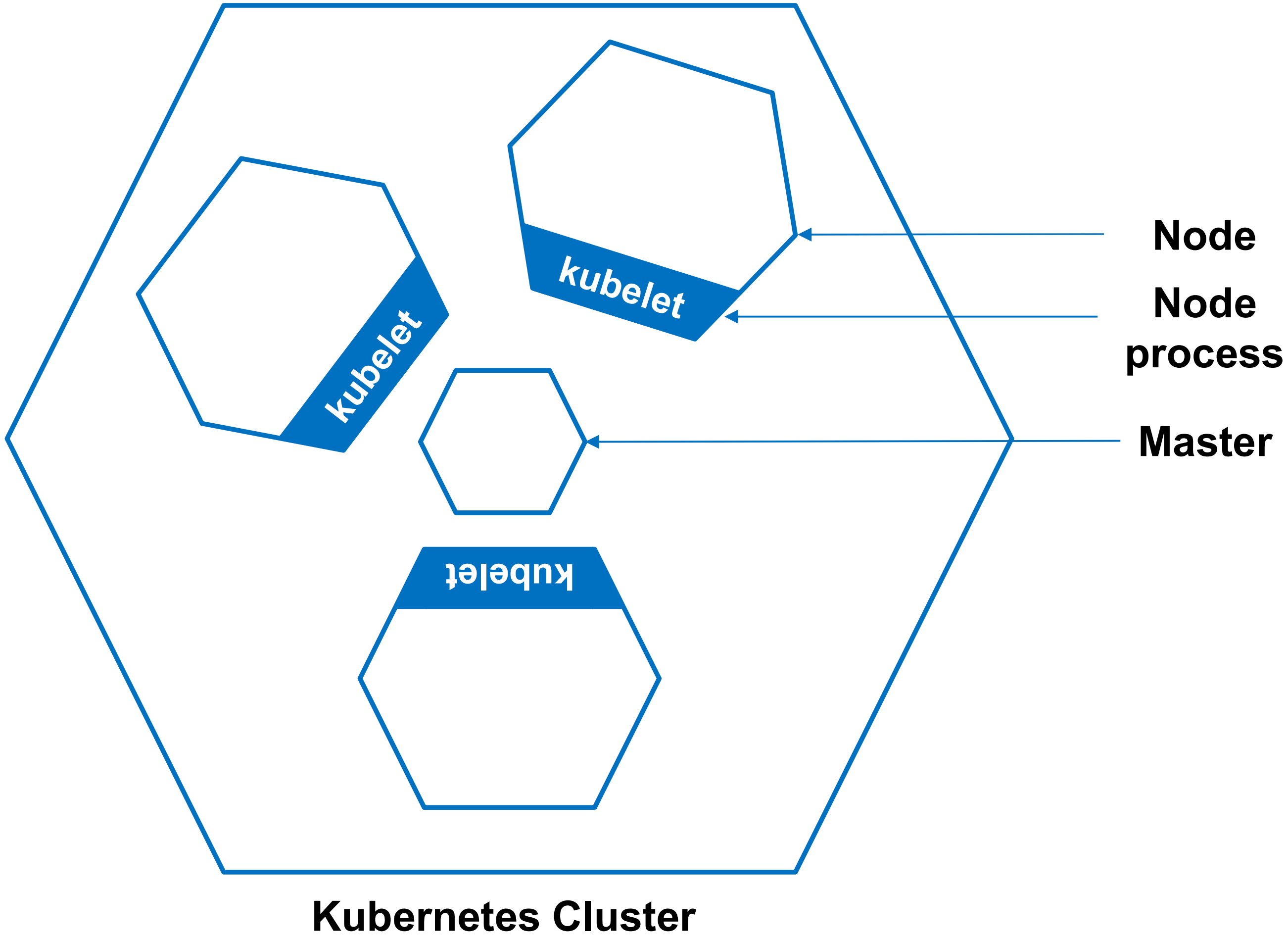
- ❑ Local development environment to run K8s
- ❑ Runs only a single VM – doesn’t provide reliability property and other features compared to a distributed cluster
- ❑ Cool for experimentation and learning ☺
- ❑ Look at the setup page – also additional information about the concepts is provided there

https://github.com/johannes-manner/k8s-soa-example

Take sometimes minutes for startup
Keep calm ☺

https://minikube.sigs.k8s.io/docs/start/

K8s CC – Master and Node



[Adapted from https://kubernetes.io/docs/tutorials/kubernetes-basics/](https://kubernetes.io/docs/tutorials/kubernetes-basics/)

K8s CC – Master and Node

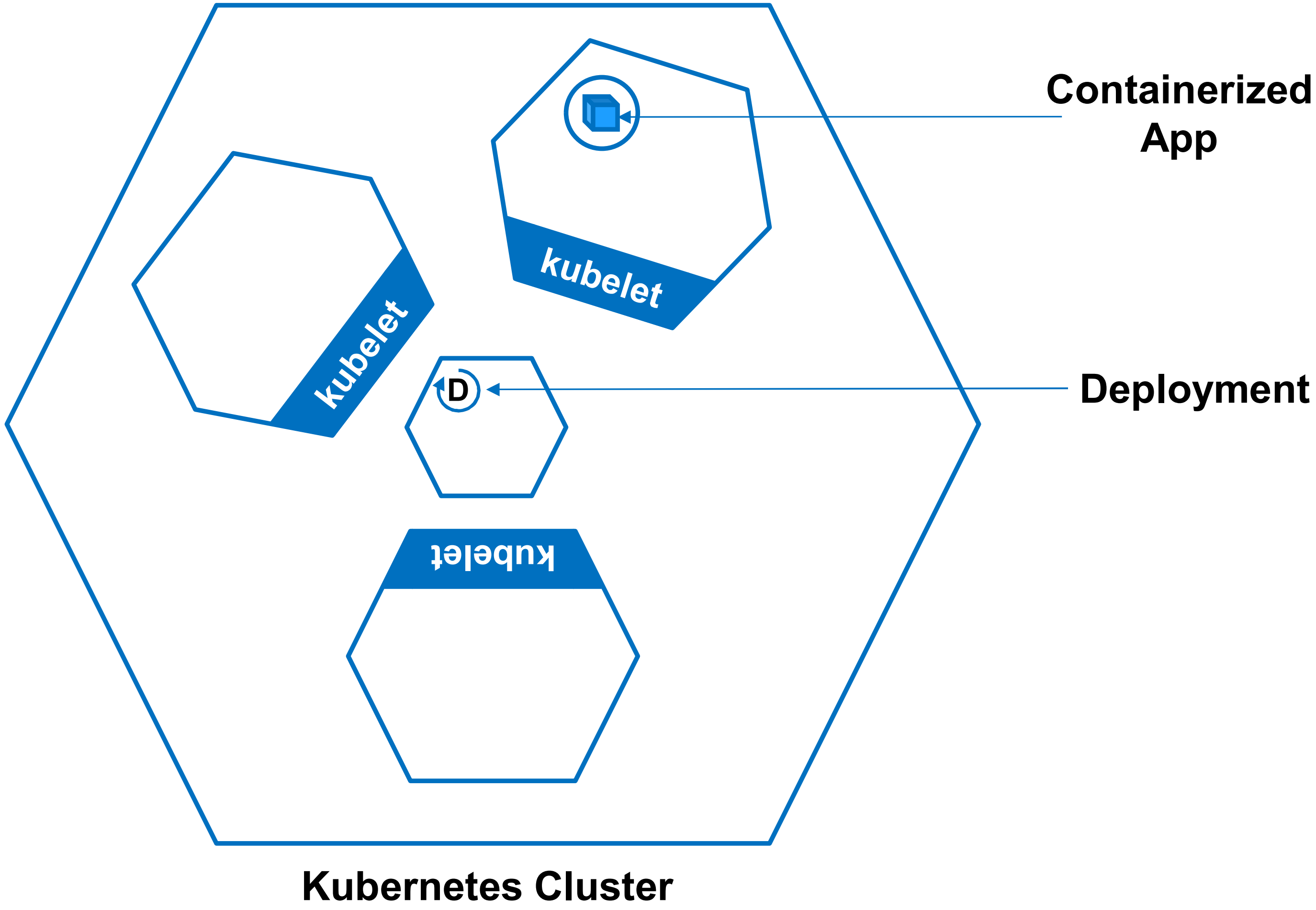
Kubernetes cluster – two types of resources:

- ❑ Master
  - Managing the cluster
  - Coordinates scheduling, maintaining desired state, scaling, rolling out new updates etc.

- ❑ Node
  - VM or physical computer
  - Serves as a worker within the cluster

- ❑ Kubelet
  - Agent for managing node
  - Node process to communicate with the master (Kubernetes API)

K8s CC – Deployment and App



Adapted from https://kubernetes.io/docs/tutorials/kubernetes-basics/

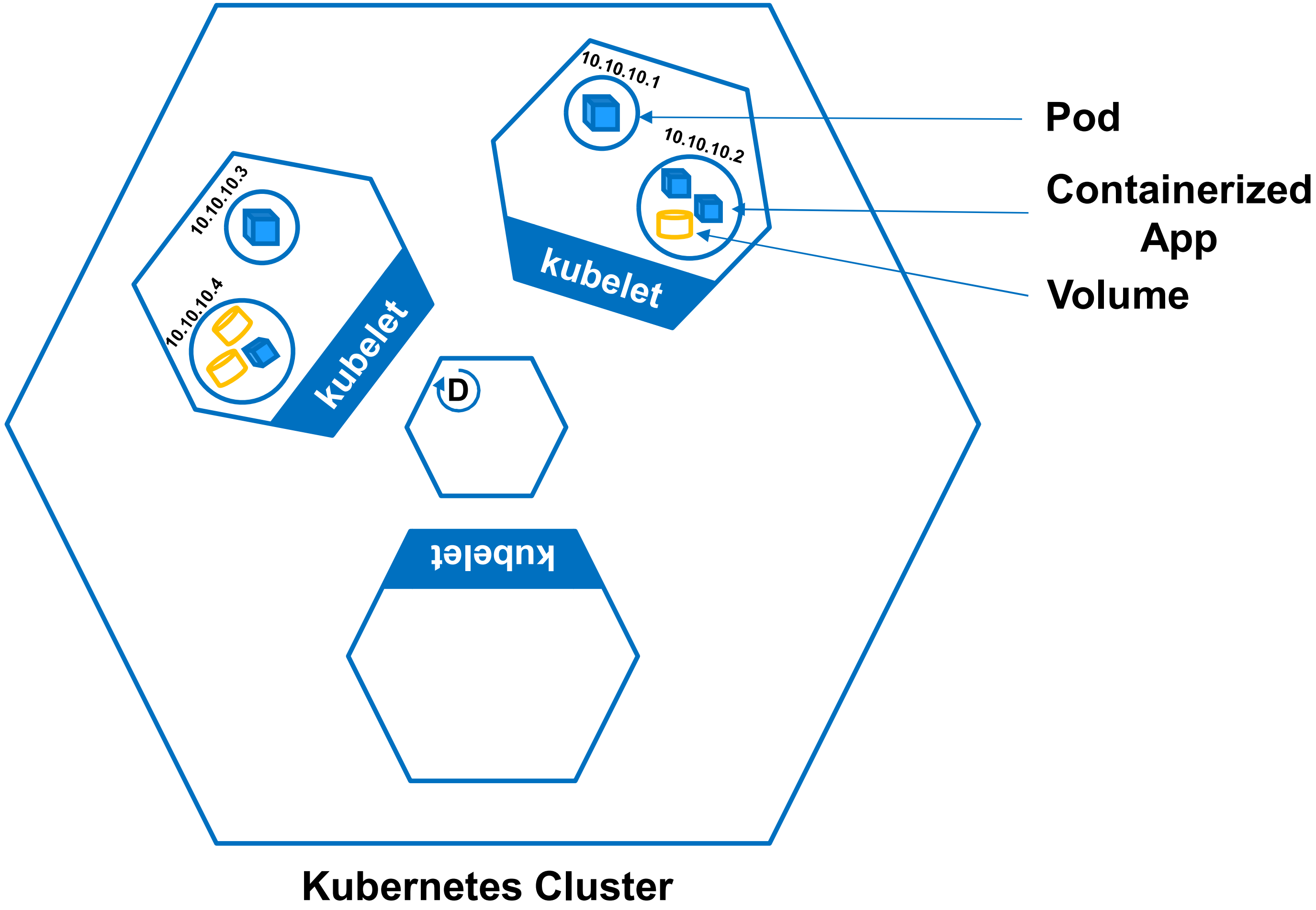
K8s CC – Deployment and App

- ❑ Deployment
  - Instruction what to create and update
  - After deployment creation, master schedules apps on nodes
  - Deployment Controller compares the desired state (declarative config) with the observed state and takes actions if necessary
    -> self-healing if machine failures occur
  - Creating and updating deployments via `kubectl` (K8s CLI)

```
$ kubectl create deployment DEPLOYMENT-NAME
            --image=ADDRESS …
```

  - Proxy to connect to the private network via `$kubectl proxy`
    (shows management information via a REST based API)
- ❑ App
  - Has to be containerized (see Docker slides)
  - Number of Replicas, ideally 3 (why?), and other settings can be specified/changed via `kubectl` commands

K8s CC – Pods



Adapted from https://kubernetes.io/docs/tutorials/kubernetes-basics/

K8s CC – Pods

- **Smallest deployable unit of K8s**
- Abstraction for a group of application containers and volumes accessible via a unique IP
- Pod IP is not exposed outside the cluster
  For debugging, use port forwarding capability of K8s
- Each container in a pod runs in its own cgroup
  Process health checks are defined per container
  (executed every 10s, if failing 3 consecutive times – container restart)
- A pod does not move, once scheduled
  (solution: destroy or reschedule the pod)
- Resources are requested per container
- Lifecycle considerations: https://dzone.com/articles/kubernetes-lifecycle-of-a-pod

K8s CC – Pods Declarative Configuration

kuard-pod.yaml

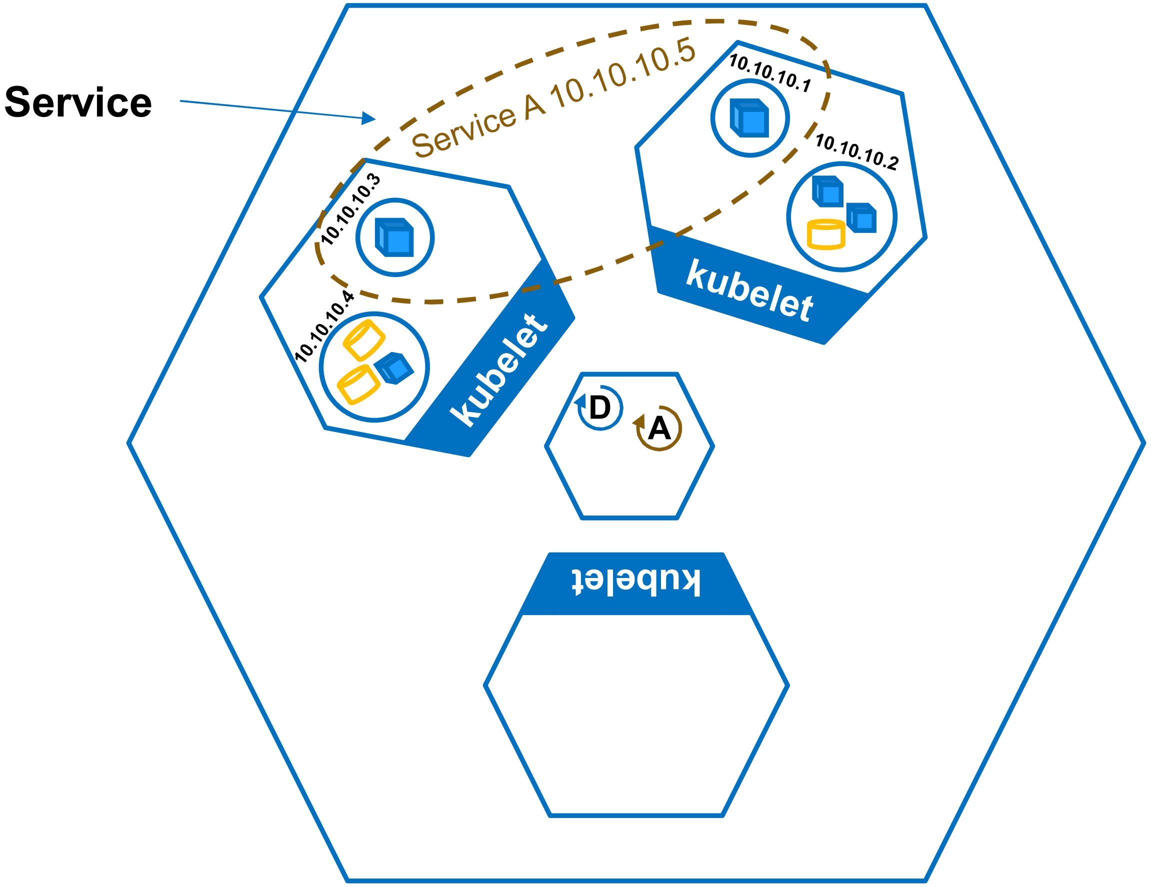
```
apiVersion: v1
kind: Pod
metadata:
  name: kuard
spec:
  containers:
  - image: gcr.io/kuar-demo/kuard-amd64:1
    name: kuard
    ports:
    - containerPort: 8080
      name: http
      protocol: TCP
```

```
$kubectl apply –f kuard-pod.yaml
```

The port 8080 is only visible within the K8s cluster, exposing the port via `$kubectl port-forward POD-NAME SERVICE-PORT:POD-Port`, in our example
`$kubectl port-forward kuard 8989:8080`
Access the pod via localhost:8989

[HBB2017, p.41]

K8s CC – Service



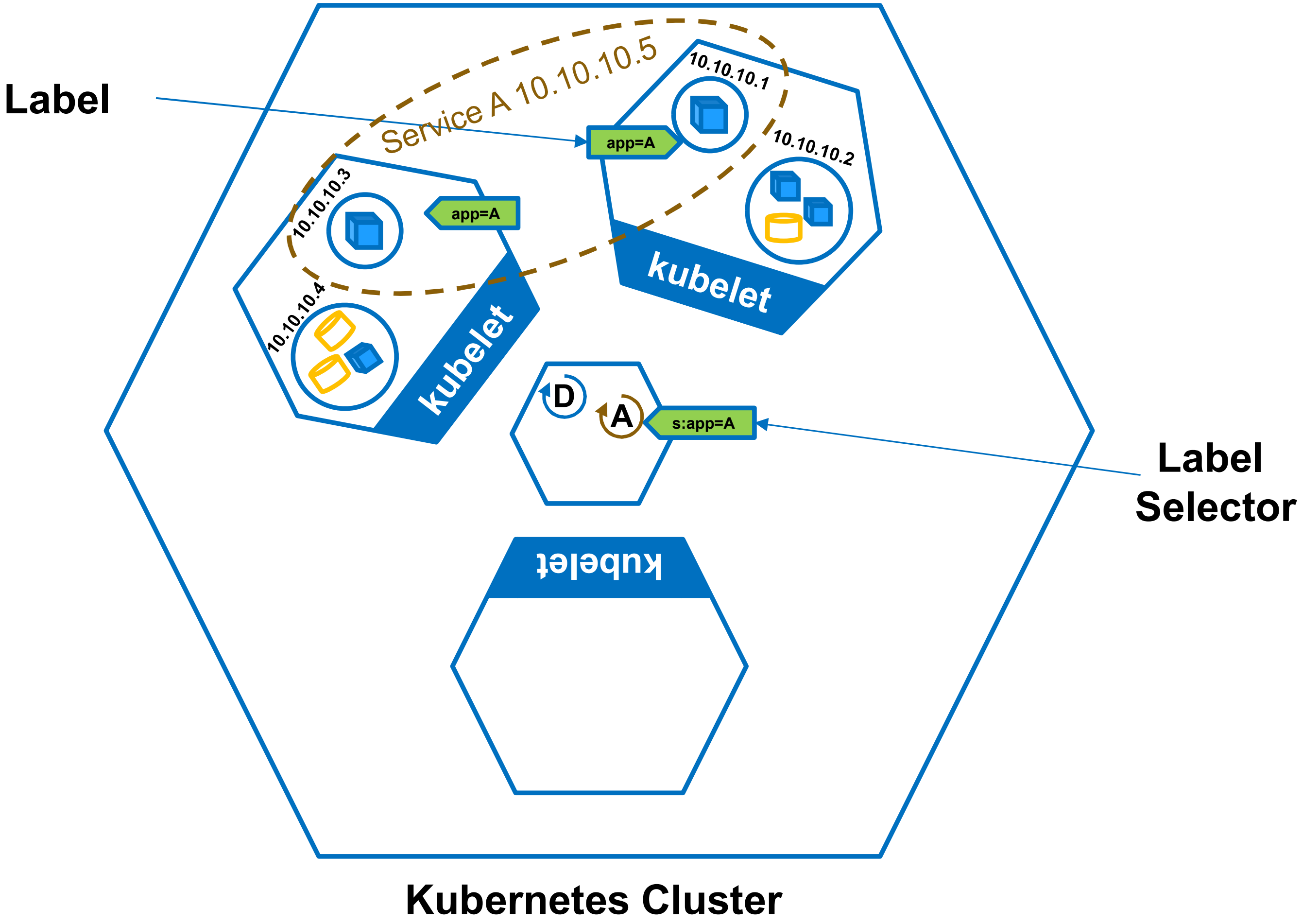
Kubernetes Cluster

Adapted from https://kubernetes.io/docs/tutorials/kubernetes-basics/

K8s CC – Service

- Service
  - Abstraction for a logical set of pods with an own IP; enable self-healing
  - Set of pods is determined by a LabelSelector
  - Enables applications to receive traffic
  - Types to expose a service
    - Cluster IP (default): service only reachable within the cluster
    - NodePort: exposes the service externally on each selected node with the same port (only for debugging useful)
    - LoadBalancer (our recommended way): Creates a load balancer and assigns a fixed, external reachable IP to it
    - ExternalName: Exposes service with an external name (kube-dns)

K8s CC – Labels and Selectors



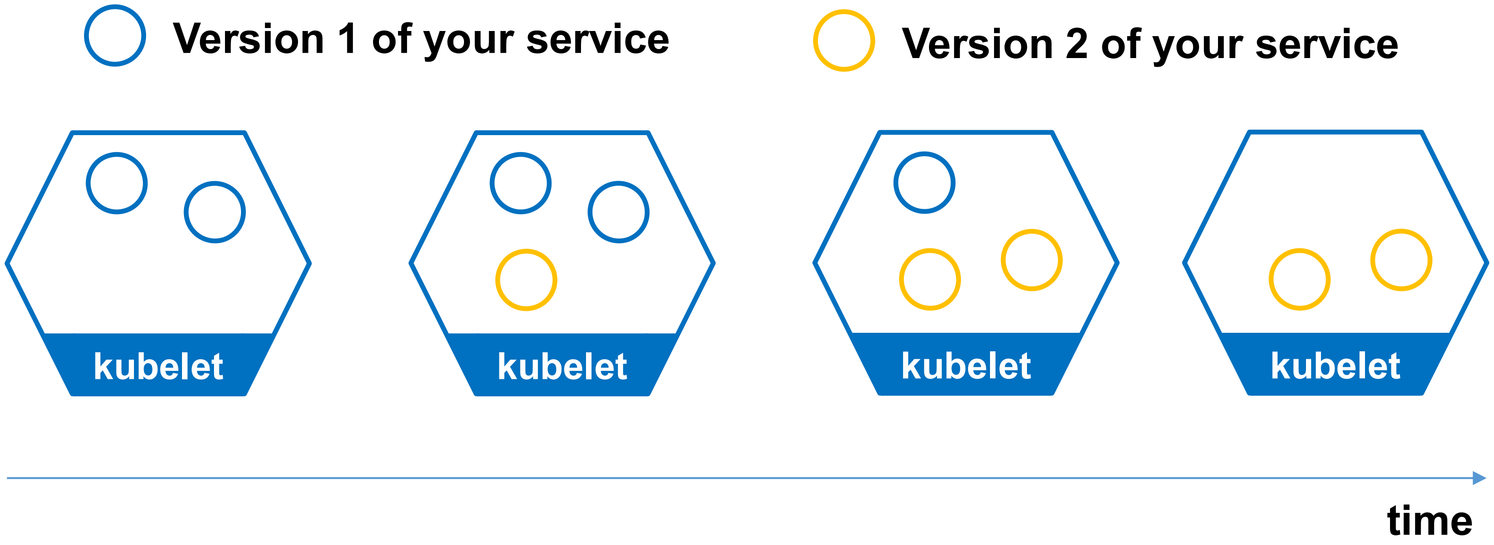
Adapted from https://kubernetes.io/docs/tutorials/kubernetes-basics/

K8s CC – Labels and Selectors

- Labels – foundation for grouping objects
  - Are key/value pairs to add metadata to your objects
  - “tier”: “frontend” – “tier”: “backend” are good examples
  - Identifying metadata - Many objects can use the same label names (grouping, viewing, operating)
- Annotations
  - Nonidentifying information for tools, libraries or as documentation
- Label Selectors
  - Group, filter a set of objects
  - Types of selectors: equality-based and set-based selectors
    - Equality-based: == and !=, e.g. environment == production
      Concatenation via “,”, e.g. a=b,c=d (means: select all objects with key/value pairs a/b and c/d)
    - Set-based : allow filtering via a set of keys and values
  - Pods of a service share the same label(s)
  - ReplicationController also uses selectors to handle pods

K8s CC – Scaling and rolling updates

- Scaling to zero and autoscaling is supported
- Services use an integrated load balancer for request distribution
- With more than one pod, you can perform rolling updates with zero downtime



K8s CC – Further Concepts

- Namespaces
  - Used for organizing objects in the cluster
  - Level of strong separation
- “IP for everything within the cluster”
  - Kubectl makes HTTP requests to access K8s objects
- Declarative configuration
  - Objects are represented as JSON or YAML files
  - Creating an object: $kubectl apply –f obj.yaml
  - Deleting an object: $kubectl delete –f obj.yaml
  - YAML is preferred since it is more human readable and allows adding comments

K8s CC – ReplicaSets

- ❑ Running multiple replicas of the same container, reasons:
  - Redundancy (Avoiding outage of the service)
  - Scaling (Handling more or less requests dynamically)
  - Sharding (Partitioning of tasks in parallel)
- ❑ Treating a set of pods as logical entity
- ❑ Basis for self-healing property of K8s since pods are rescheduled under specific failure conditions
- ❑ **Replication Controller** compares desired and actual state and takes action if necessary, e.g. start /delete a pod
- ❑ Monitoring is done by labels, a ReplicaSet knows its pods via the same labels

K8s CC – DaemonSets

- Ensures a single copy on every node in the K8s cluster
- Examples: Logging and/or monitoring agents
- Managed by a replication controller

Difference to ReplicaSets

- Multiple replicas managed by a ReplicaSet can be placed on the same node, whereas DaemonSets schedule a single instance on each node in the cluster

K8s CC – Self healing property - Example

```
$ kubectl apply -f cats-rs.yaml
```

Created a ReplicaSet with 3 replicas

```
$ kubectl get pods
Unable to connect to the server: dial tcp 192.168.99.107:8443: connectex:
```

Minikube crashed

```
$ kubectl get pods
NAME              READY    STATUS              RESTARTS        AGE
cats-rs-jh8hm     0/1      Error               1               3m2s
cats-rs-qcstl     0/1      Error               2               3m2s
cats-rs-tgvjl     0/1      Error               1               3m2s
```

No Pods are ready to serve requests

```
$ kubectl get pods
NAME              READY    STATUS              RESTARTS        AGE
cats-rs-jh8hm     1/1      Running             2               3m15s
cats-rs-qcstl     0/1      CrashLoopBackOff    2               3m15s
cats-rs-tgvjl     0/1      Error               2               3m15s

$ kubectl get pods
NAME              READY    STATUS              RESTARTS        AGE
cats-rs-jh8hm     1/1      Running             2               3m28s
cats-rs-qcstl     0/1      CrashLoopBackOff    2               3m28s
cats-rs-tgvjl     0/1      CrashLoopBackOff    2               3m28s
```

Self healing (replication controller) of K8s starts/reschedules new pods

```
$ kubectl get pods
NAME              READY    STATUS              RESTARTS        AGE
cats-rs-jh8hm     1/1      Running             2               3m37s
cats-rs-qcstl     1/1      Running             3               3m37s
cats-rs-tgvjl     1/1      Running             3               3m37s
```

K8s CC – Data Integration

- ❑ Often the most complicated step in building distributed systems in container and container orchestration environments
- ❑ Due to legacy systems and evolution of the cloud, data services are often externalized cloud services
  -> difficult/impossible to host them in a local cluster

- ❑ Two Options for data integration in K8s
  1. Externalized cloud service like DynamoDb (default for many enterprise scenarios – think about ease of use)
  2. Using storage solutions within K8s
     a) Running Reliable Singletons
     b) Using StatefulSets

K8s CC – DI – 2a) Reliable Singletons

- ❑ Use K8s features like ReplicaSets, BUT not replicate the storage
- ❑ As reliable as running a single database VM or physical server, i.e. more reliable since replication controller handles failures or other outages of your singleton
  -> quite simple (no storage replication)
  -> only short downtime (for many systems acceptable)

K8s CC – DI – 2b) StatefulSets

Difference to ReplicaSets

- ❑ In a ReplicaSet, all pods are homogeneous (can be replaced without any action from the K8s orchestrator) – Pods in a StatefulSet have the following properties:
  - Each replica gets a unique index and name(e.g., mongodb-0, mongodb-1 etc.)
  - Creation is sequentially and blocks until the previous one is healthy and running
  - Deletion of pods is the other way around; starting from the highest index
- ❑ The service for a StatefulSet is “headless”, means that the service has no ClusterIP since all pods are unique and have its own access point

K8s CC – DI – 2b) StatefulSets and Volumes

- ❑ Replicate the stateful layer:
  Specify a primary replica (normally pod xy-0) and add all the other database replicas to the primary replica for synchronizing state
- ❑ Persistent storage:
  Mount a volume into a specific folder of your pod (in mongoDB into /data/db)

- ❑ For each created database replica, a single volume is created too, so be careful with consistency (read the docs of the used databases)

Outlook – what’s missing right now

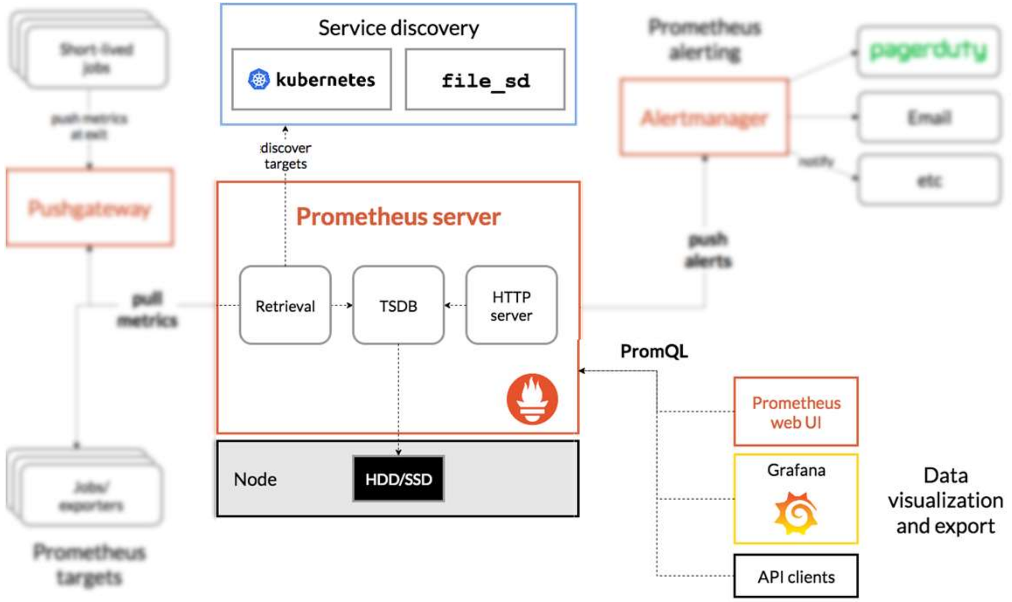
- ❑ Monitoring and Health Checks
- ❑ Persistent Volumes
- ❑ Jobs (third form of replicated pods)
- ❑ ConfigMaps
- ❑ Secrets
- ❑ …

Monitoring

- ❑ Monitoring and getting information is not that easy in a containerized application (all entities are encapsulated)
- ❑ Use Pod's container information via:
  - $ kubectl describe pod <pod-name>
  - $ kubectl logs <pod-name> -c <container-name>
- ❑ Specific information can be found in K8s docu: https://kubernetes.io/docs/tasks/debug-application-cluster/
- ❑ Tools for Monitoring: Prometheus (https://prometheus.io/)
  - a multi-dimensional data model with time series data identified by metric name and key/value pairs
  - no reliance on distributed storage; single server nodes are autonomous
  - multiple modes of graphing and dashboarding support

Tool info copied from: https://prometheus.io/docs/introduction/overview/

Monitoring: Prometheus Architecture



- **State of the art:** Using a combination of Kubernetes for container orchestration, Prometheus for collecting metrics and Grafana for visualizing this data

Health Checks

- ❑ K8s checks the health of your containers via “process health checks” permanently
- ❑ If container fails this check, K8s restarts it.
- ❑ Checks are application specific and included in YAML files

Liveness

- Specify endpoint* for liveness probes
- Timeout, interval and threshold
- If container does not respond in appropriate time, container is restarted (that’s the default; dependent on restart policy)

Result: application is running properly

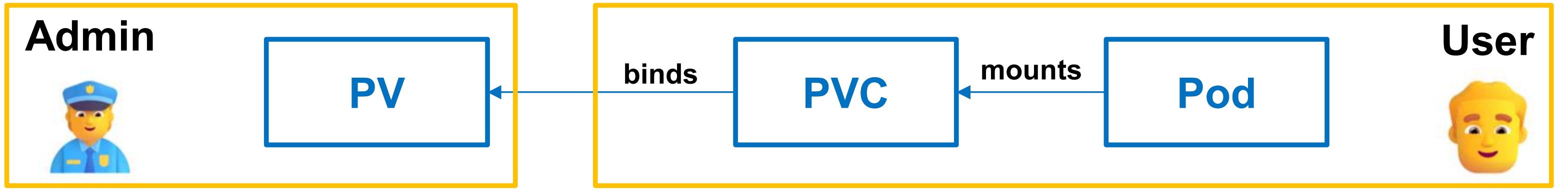
Readiness

- Specify endpoint* for readiness probes
- Timeout, interval and threshold
- If container does not respond in appropriate time, container is removed from service load balancers

Result: application is ready to serve requests

* Endpoints can be a HTTP endpoint (e.g. for microservices) or tcpSocket (e.g. for databases), dependent on the service.

Persistent Volumes Subsystem



- “The PersistentVolume subsystem provides an API for users and administrators that abstracts details of how storage is provided from how it is consumed”
  - PersistentVolume (PV):
    - Created by a cluster administrator
    - Piece of Storage in your cluster (handled like any other Kubernetes object)
    - Lifecycle independent of individual pods
  - PersistentVolumeClaim (PVC):
    - Request for storage by a user
    - PVCs consume PV resources
    - Claims can access specific size and access modes on a PV (decoupling)
- Control loop in master watches for PVCs and binds them to matching PVs
- Pods use claims as volumes. Per default the master looks dynamically for a matching PV. (Use label selectors)

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Jobs – Ephemeral Task Executions

- Runs a specific task to completion
  No pods are running when the task is done
- Creates pods as specified (also restarts pods of jobs, which are not completed yet)
- When job completes, pods are not deleted (kept for investigation, i.e. container logs)
- Deleting the job also deletes the associated pods
- You can also define periodic jobs (cron jobs)

https://kubernetes.io/docs/concepts/workloads/controllers/job/
https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/

ConfigMaps and Secrets

- ConfigMaps: Set of external variables for container
- Secrets: For sensitive environment variables/data

- Three ways of using a ConfigMap
  - Filesystem: mount ConfigMap in a Pod
  - Environment variable: set values of environment variables
  - CLI argument: K8s creates CLI argument based on ConfMap

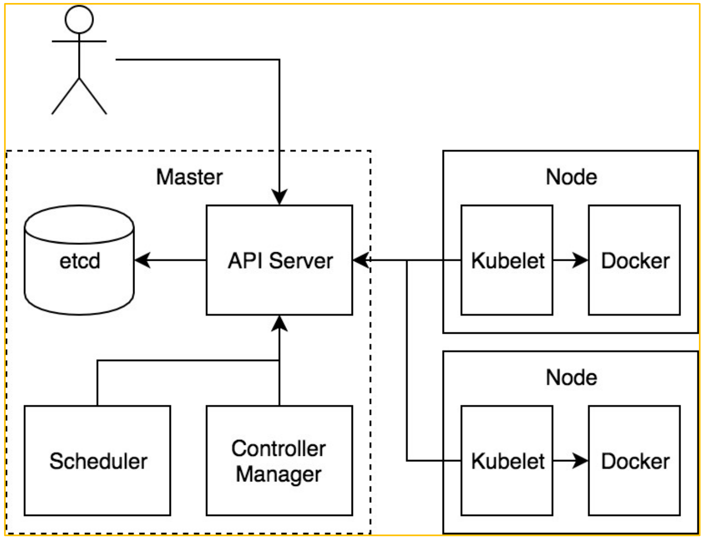
ConfigMap YAML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: special-config
  namespace: default
data:
  serverUri: very-best
```

Part of Pod’s YAML

```
env:
  - name: DEMO
    value: "Test it"
  - name: SERVER_URI
    valueFrom:
      configMapKeyRef:
        name: special-config      # name of the config map
        key: serverUri            # key of the config property
```

Wrap up: K8s System Architecture



Components

etcd: the core state store for Kubernetes (hidden, only accessible via API Server!)

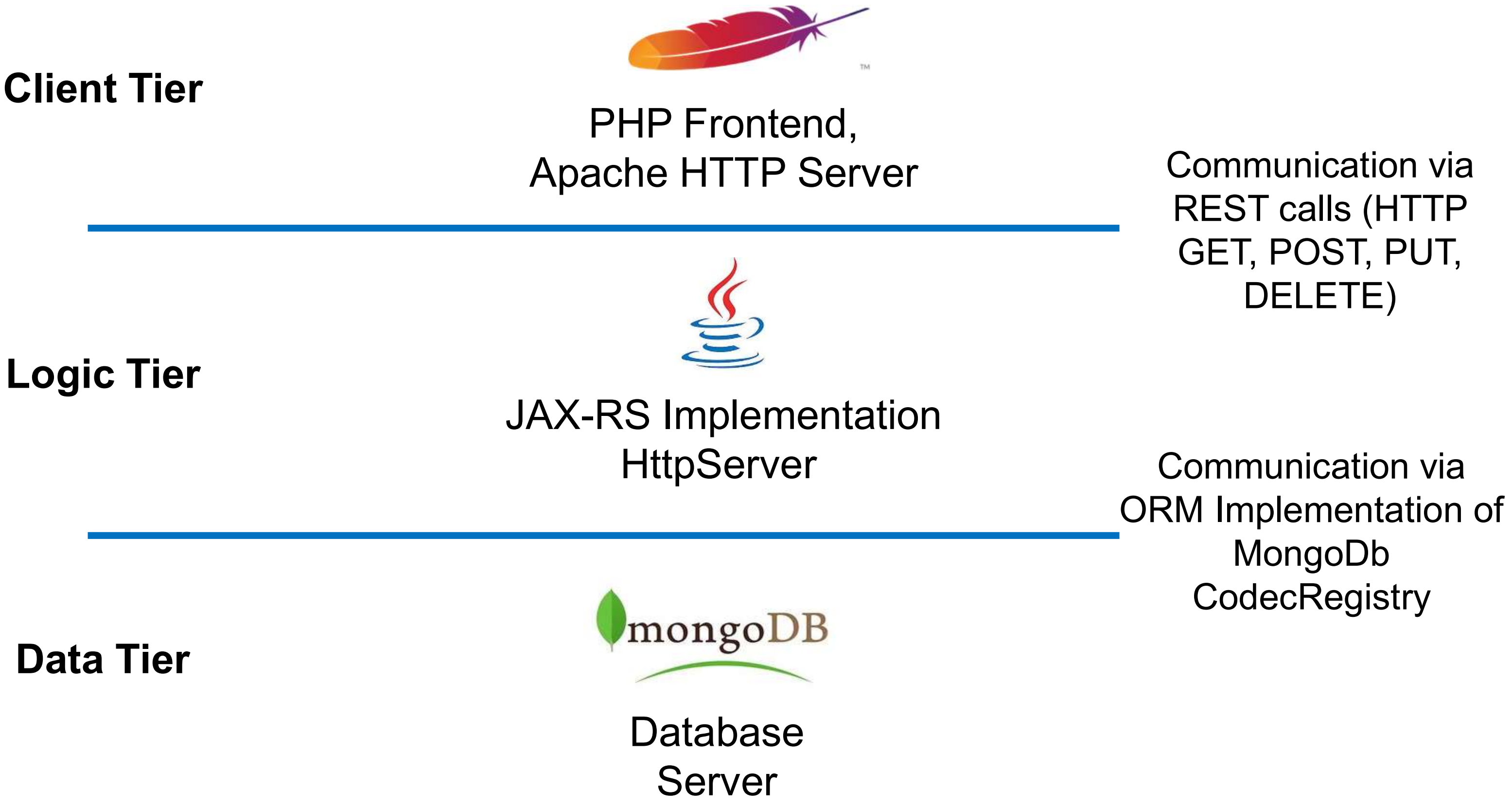
API Server: Simple K8s REST API Responsible for Authentication/ Authorization, admission controller (reject or adjust requests), API versioning.

Scheduler/Controller Manager: makes system work: bind and unbind Pods to nodes. Place of the reconciliation controller, etc.

Kubelet: K8s agent on every node

https://blog.heptio.com/core-kubernetes-jazz-improv-over-orchestration-a7903ea92ca

Use Case: Cat Service - Architecture



Client Tier

PHP Frontend,
Apache HTTP Server

Communication via REST calls (HTTP GET, POST, PUT, DELETE)

Logic Tier

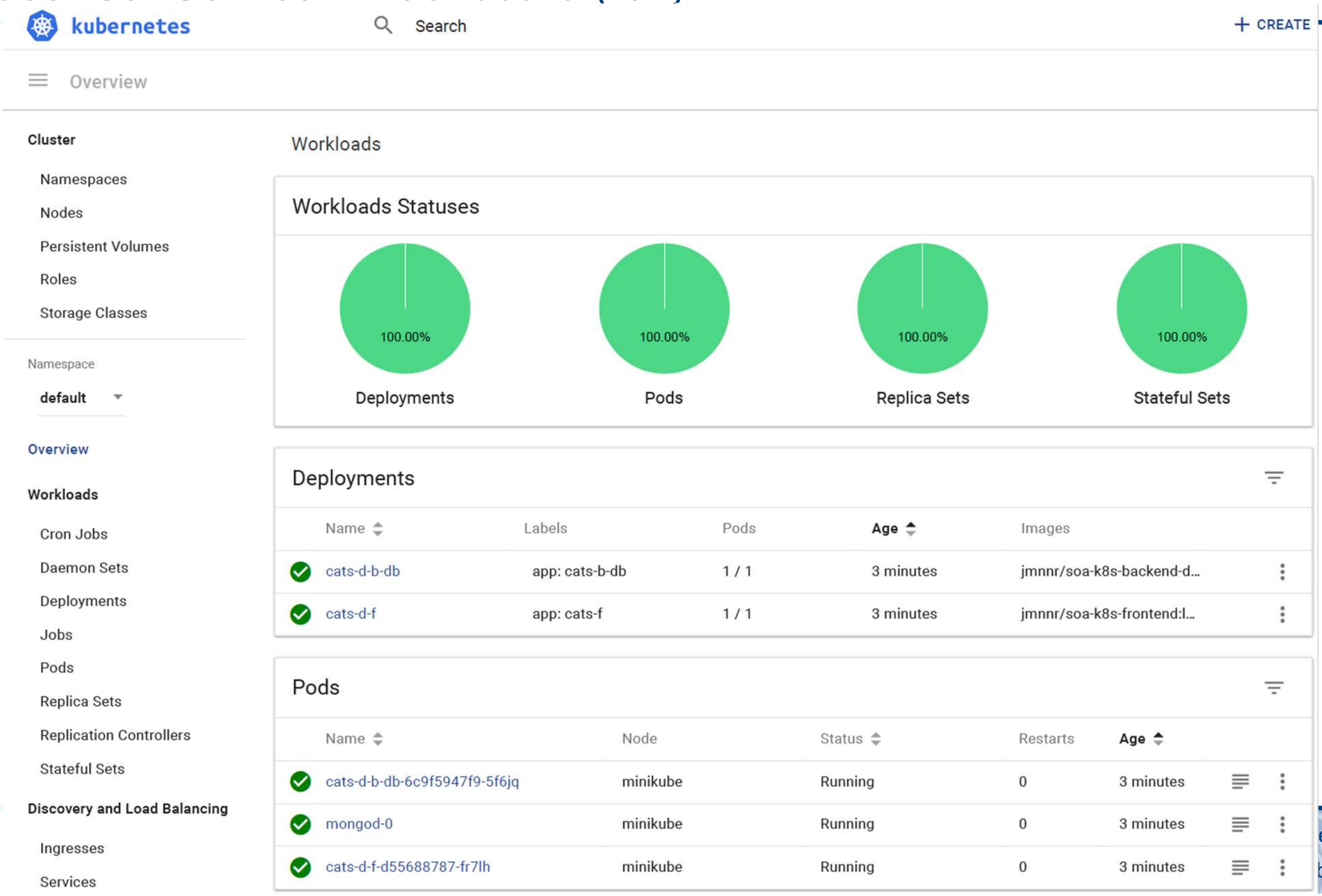
JAX-RS Implementation
HttpServer

Communication via ORM Implementation of MongoDb CodecRegistry

Data Tier

Database
Server

Use Case: Cat Service – Dashboard (1/2)



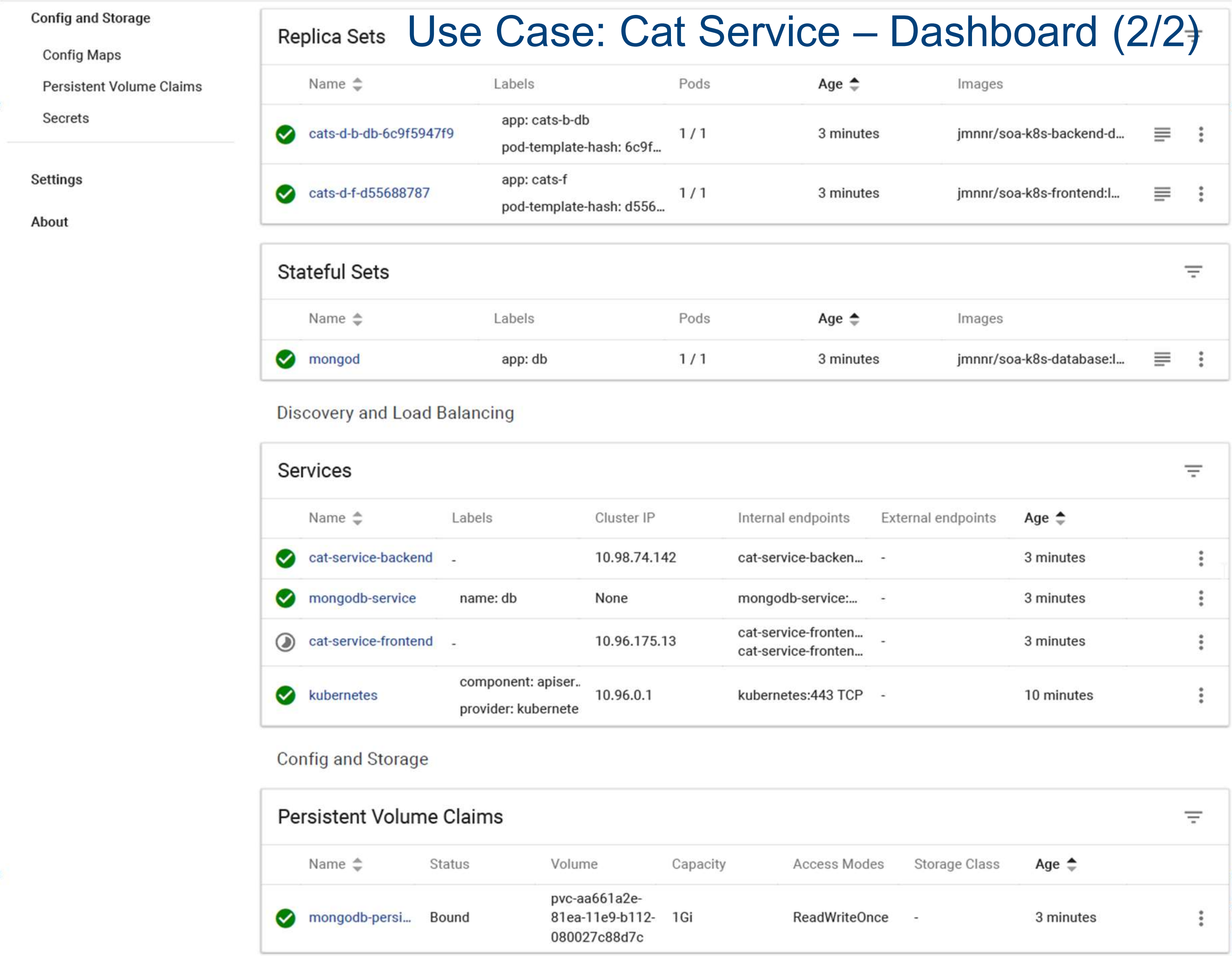
Deployments

	Name	Labels	Pods	Age	Images
✓	cats-d-b-db	app: cats-b-db	1 / 1	3 minutes	jmnnr/soa-k8s-backend-d...
✓	cats-d-f	app: cats-f	1 / 1	3 minutes	jmnnr/soa-k8s-frontend:l...

Pods

	Name	Node	Status	Restarts	Age
✓	cats-d-b-db-6c9f5947f9-5f6jq	minikube	Running	0	3 minutes
✓	mongod-0	minikube	Running	0	3 minutes
✓	cats-d-f-d55688787-fr7lh	minikube	Running	0	3 minutes

Use Case: Cat Service – Dashboard (2/2)



Replica Sets

	Name	Labels	Pods	Age	Images
✓	cats-d-b-db-6c9f5947f9	app: cats-b-db pod-template-hash: 6c9f...	1 / 1	3 minutes	jmnnr/soa-k8s-backend-d...
✓	cats-d-f-d55688787	app: cats-f pod-template-hash: d556...	1 / 1	3 minutes	jmnnr/soa-k8s-frontend:l...

Stateful Sets

	Name	Labels	Pods	Age	Images
✓	mongod	app: db	1 / 1	3 minutes	jmnnr/soa-k8s-database:l...

Discovery and Load Balancing

Services

	Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
✓	cat-service-backend	-	10.98.74.142	cat-service-backen...	-	3 minutes
✓	mongodb-service	name: db	None	mongodb-service:...	-	3 minutes
◔	cat-service-frontend	-	10.96.175.13	cat-service-fronten... cat-service-fronten...	-	3 minutes
✓	kubernetes	component: apiser.. provider: kubernete	10.96.0.1	kubernetes:443 TCP	-	10 minutes

Config and Storage

Persistent Volume Claims

	Name	Status	Volume	Capacity	Access Modes	Storage Class	Age
✓	mongodb-persi...	Bound	pvc-aa661a2e-81ea-11e9-b112-080027c88d7c	1Gi	ReadWriteOnce	-	3 minutes

Literature – Interesting sources

- [BGOBW2016] B. Burns et al., “Borg, Omega, and Kubernetes,” ACM Queue, vol. 14, pp. 70–93, 2016
- [HBB2017] K. Hightower, B. Burns, and J. Beda,: Kubernetes: Up and Running. O’Reilly UK Ltd., 2017.
  German version: https://katalog.ub.uni-bamberg.de/query/BV044795699
- Nigel Poulton: The Kubernetes Book (e-Book, frequently updated, but not that mature as “up and running”)
- https://kubernetes.io/docs/tutorials/#basics

Second edition available

- CatService UseCase: https://github.com/johannes-manner/k8s-soa-example