Kubernetes Training

Introduction

Organization

- Labs
  - 3 labs during this training
  - Done by everyone in shared Kubernetes cluster setup for this training
  - Everyone has admin rights to this cluster
  - Please keep within your namespace
  - Don‘t test the limits of the cluster, we don‘t want to spent half of the training repairing it ☺
- All labs and demos as well as these slides are on GitHub:
  - https://github.com/scholzj/dbg-kubernetes-training

What is Kubernetes

Kubernetes

- From Greek
  - Means helmsman / ship‘s pilot
  - Steers your ship full of containers ☺
  - Kubernetes = k8s
- System for automated deployment, scaling and management of containerized applications
- „A true cloud platform“
- Open source (ASL 2.0), governed by Cloud Native Computing Found.
- Planet scale, flexible, runs everywhere
- Written in Go

Containers

- Containers are key to application deployments in Kubernetes
- They are the only supported type of workload
- Kubernetes supports different container runtimes
  - Container Runtime Interface = pluggable mechanism for container runtimes
  - Docker
  - rkt / rktlet
  - cri-o / OCI containers (through CRI)
  - Frakti containers (hypervisor based containers)
- Containers allow to run heterogeneous workloads
  - Everything what can be packed into container can run on Kubernetes

Main features

- Self-healing
  - Restarts applications when they fail
  - Kills containers which do not respond
- Horizontal scaling
  - Allows scaling of applications up and down
- Automatic binpacking
  - Schedules the containers according to different requirements (density, reliability)
- Secrets and Configuration management
  - Deploys and manages secrets and configurations

Main features

- Service discovery and load balancing
  - Find applications using their DNS names
  - Load balance across set of containers
- Rollouts and rollbacks
  - Progressively deploy changes or roll them back in case of problems
- Storage orchestration
  - Create storage on supported platforms (AWS, GCP, Gluster, Ceph, …)
- Batch execution
  - Can run long running services but also batch jobs

History

- Google has 15 years of experience with running production workloads in containers and had a major role in container development
- „Large-scale cluster management at Google with Borg“
- „Borg, Omega and Kubernetes“
- Kubernetes are not open source version of Borg or Omega, but …
  - They are built by the same people
  - They build on the experience from Borg and Omega
  - Take the good things, improve the bad things
- Other frameworks building on Borg heritage (Mesos, CloudFoundry)

Why should you use Kubernetes

- Industry standard for containerized deployments
- Higher computational density
  - Better HW utilization
  - Lower costs
- Abstract infrastructure details from applications
  - Deployed applications don’t care whether the cluster runs on-prem or with different cloud providers
- Make writing and running cloud native applications easier
- Supports many different types of workloads (long running services, batch jobs, etc.)

Distributions

- Kubernetes are open source
- Many companies offer „distributions“ of Kubernetes
  - Might contain enhancements and changes
  - Often simplify deployment on premise or into different public clouds
  - Red Hat OpenShift
  - Tectonic form CoreOs
  - Rancher
  - Canonical
  - VMware
  - Public cloud providers (Google Container Engine, Microsoft Container Service)

Alternatives

- Apache Mesos
- Marathon
- Apache Aurora
- Pivotal Cloud Foundry
- Docker Swarm
- HashiCorp Nomad
- Rancher

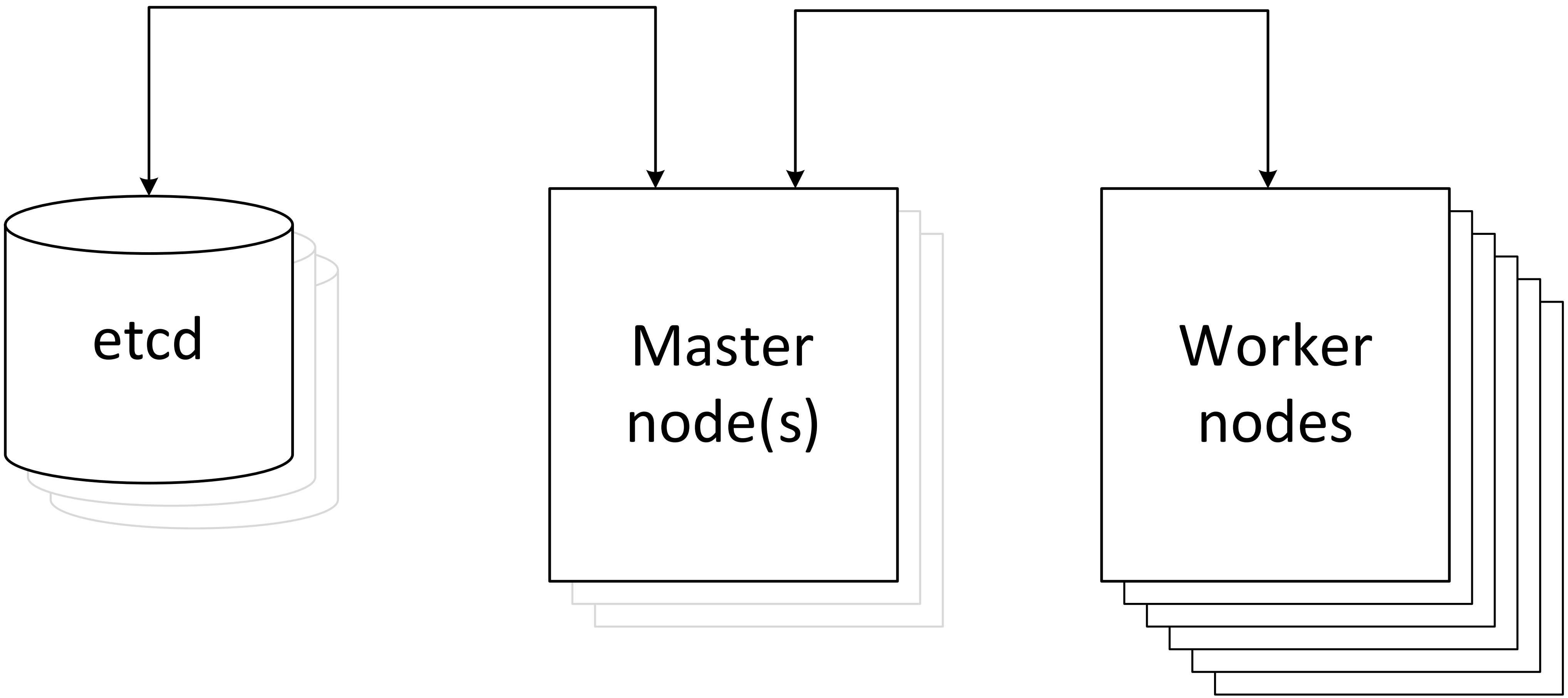
Kubernetes cluster

Kubernetes architecture

- 3 main parts
  - Master node(s) / control plane
  - etcd store
  - Worker nodes

- Components can run as regular processes (systemd units) or as Docker images

Kubernetes architecture



Master node(s)

- Runs the API server, controller manager and scheduler
- API server
  - Exposes REST API for access to Kubernetes resources
- Scheduler
  - Places containers onto worker nodes
- Controller Manager
  - Runs in loops and reconciles the actual state of the cluster with the expected state
- Either in standalone mode or in HA mode with elected leaders
- In some installations it might run also other (system) components

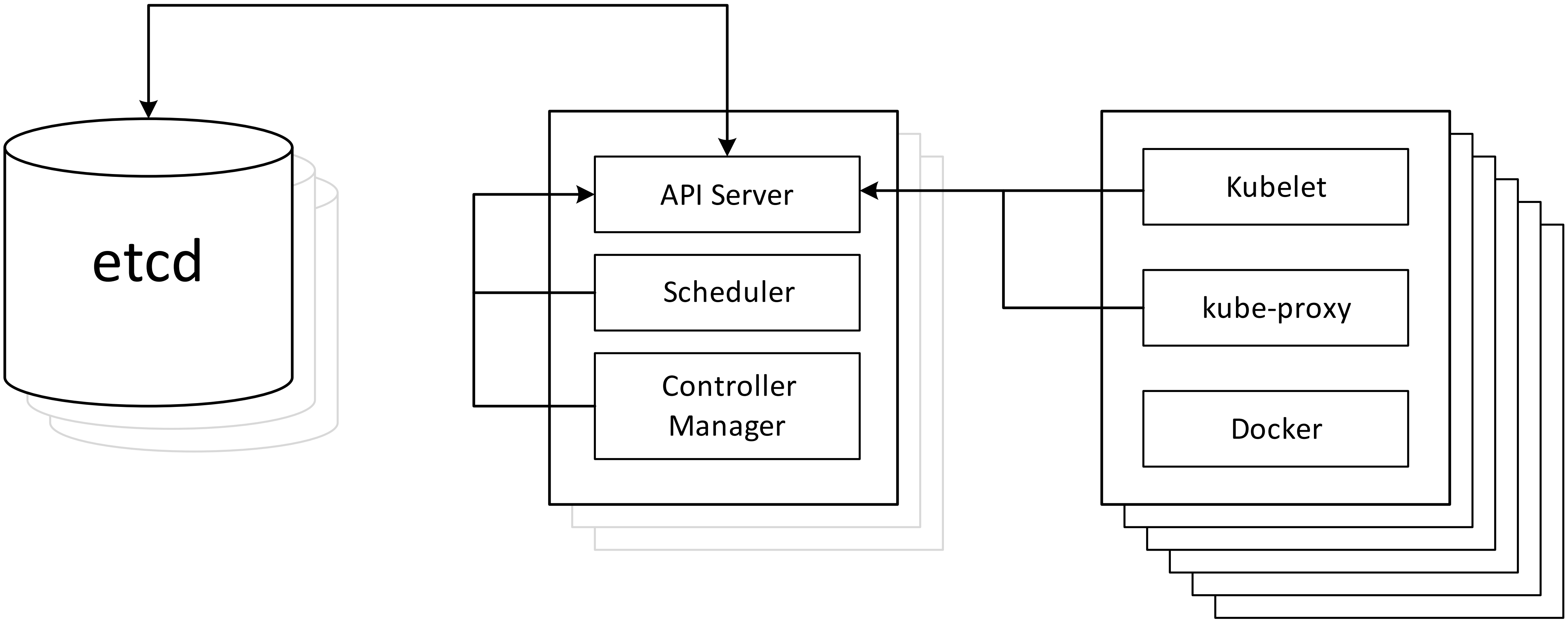
etcd

- Used to store the cluster state
- Distributed key-value store
- Data are replicated across instances
- Leader is elected from the nodes
- etcd can be used to gain access to the cluster and therefore is needs to be secured / protected
- Can run on separate hosts or on the master nodes
- Accessed only by the API manager, not by any other components or by the deployed applications

Worker nodes

- Used to run deployed applications
- Kubelet
  - Controls that the correct containers are running
- Proxy
  - Manages network connectivity between containers
- Kubelet and Proxy are running on each worker node

Kubernetes architecture



Networking

- Uses a network bridge to connect to the network
- Every computing unit (Pod) has its own IP address
  - Avoid problems with port conflicts between different computing units
  - Computing unit = Pod (group of related containers)
- The network setup can be achieved using SDN or physical network
- Several different SDNs:
  - Weave, Calico, Flannel
- Kubernetes have their own Network Policies
  - Kubernetes support only ingress network policies
  - Some SDNs like Calico support also egress network policies

Demo 1: Kubernetes components

Kubernetes installation

Installation

- Manually
  - Prepare physical / virtual machines
  - Install the Kubernetes software
  - Wire the different components together
- Different tools to simplify installation
- Often integrated directly into public clouds
- Different deployments
  - Single node deployments (for development)
  - Single node etcd, single node master and multiple workers
  - Single node etcd, HA master and multiple workers
  - HA etcd, HA master and multiple workers
- Different distributions might have their own deployment tools

Public clouds

- Google Container Engine (GKE)
  - Part of Google Cloud
  - Easiest way how to get started with Kubernetes
  - Setup the cluster with few clicks from the Cloud Console or from the gcloud command line tool
- Microsoft container Service
  - Part of Microsoft Azure cloud
- OpenShift Online
  - Based on OpenShift

Tools

- Kops
    - Cluster deployment into Amazon AWS
    - Can generate Terraform templates
    - Planned to support also other clouds (GCP)
- Kargo
    - Based on Ansible
    - Can deploy production ready cluster to bare metal and many different clouds
- kube-aws
    - Uses Amazon CloudFormation
    - Deploys Kubernetes in AWS using CoreOS
- kubeadm
    - Helps with Kubernetes installation on existing machines
    - Expected to be part of something „bigger“

Demo 2: Training cluster in AWS

Development installations

- Minikube
  - Single node installation of Kubernetes
  - Runs in VM on Linux or OS X
  - Part of Kubernetes project
- Hyperkube
  - Kubernetes in single Binary
  - Runs a kubelet and lets it deploy all other Kubernetes components

Demo 3: minikube

Add-ons

- Add-ons provide some enhanced functionality to the cluster
  - Not required, but recommended
- DNS service
- Heapster
  - Monitoring of HW resources (CPU, RAM)
  - Can forward the data into resource monitoring systems (Influx, Grafana, …)
  - Can be used for auto-scaling
- Fluentd
  - Used to collect logs from running containers
  - Can forward the logs into ElasticSearch etc.

How to try Kubernetes?

- Use Minikube on localhost
  - https://github.com/kubernetes/minikube

- Use Kops to deploy Kubernetes into our AWS Sandbox:
  - https://github.com/scholzj/aws-k8s-kops-ansible

Accessing Kubernetes

Kubernetes API

- Kubernetes provides REST API
  - API can be used to access all Kubernetes resources
- Split into API groups
  - Some resources are in the core API, e.g. api/v1
  - Some are part of beta or alpha extensions, e.g. api/extensions/v1beta1
- Every resource has
  - Kind, API version, Metadata, Spec
- Uses OpenAPI / Swagger to describe the API
- Can be accessed directly or through provided tools

Demo 4: API

kubectl

- CLI tool for working with the Kubernetes API
  - Can be used to create, list or delete resources
  - Communicates via the REST API
- Advanced usage
  - Proxy connections between cluster and localhost
  - Get latest logs
  - Connect into running containers

kubectl

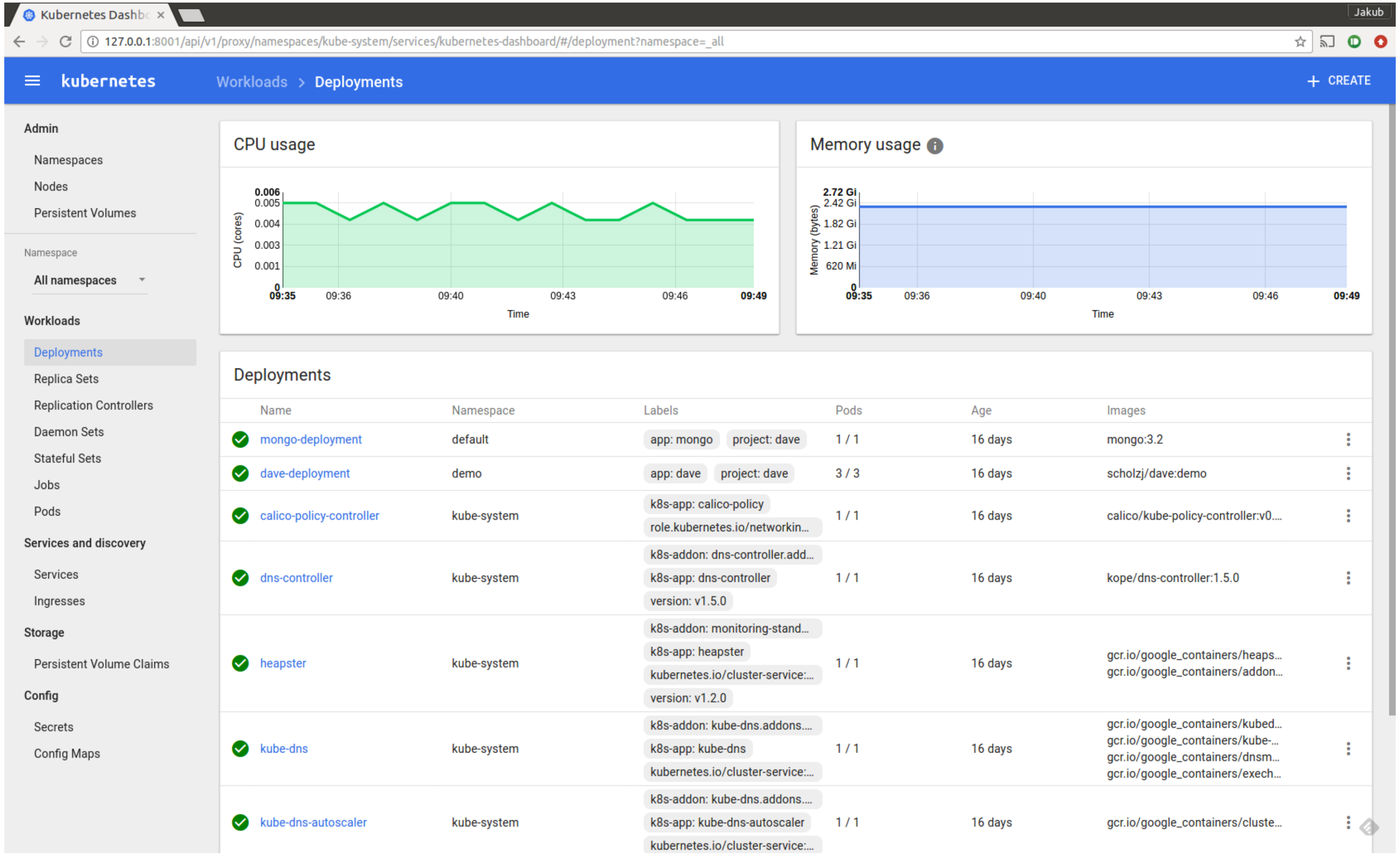
- Configuration is stored locally
  - „cluster“ configuration contains the API endpoint etc.
  - „credentials“ configuration contains the configuration of the user
  - „context“ links one cluster and one credentials object
- Current context is used as default
  - „—context“ can be used to overwrite the current context

Demo 5: kubectl

Kubernetes Dashboard

- Browser based UI
- Monitor and manage resources deployed into the cluster
- Monitor and manage the cluster it self
- Monitor resources consumed by deployed applications
- Runs as any other application deployed into Kubernetes

Kubernetes Dashboard



Demo 6: Kubernetes Dashboard

Lab 1: Accessing Kubernetes cluster

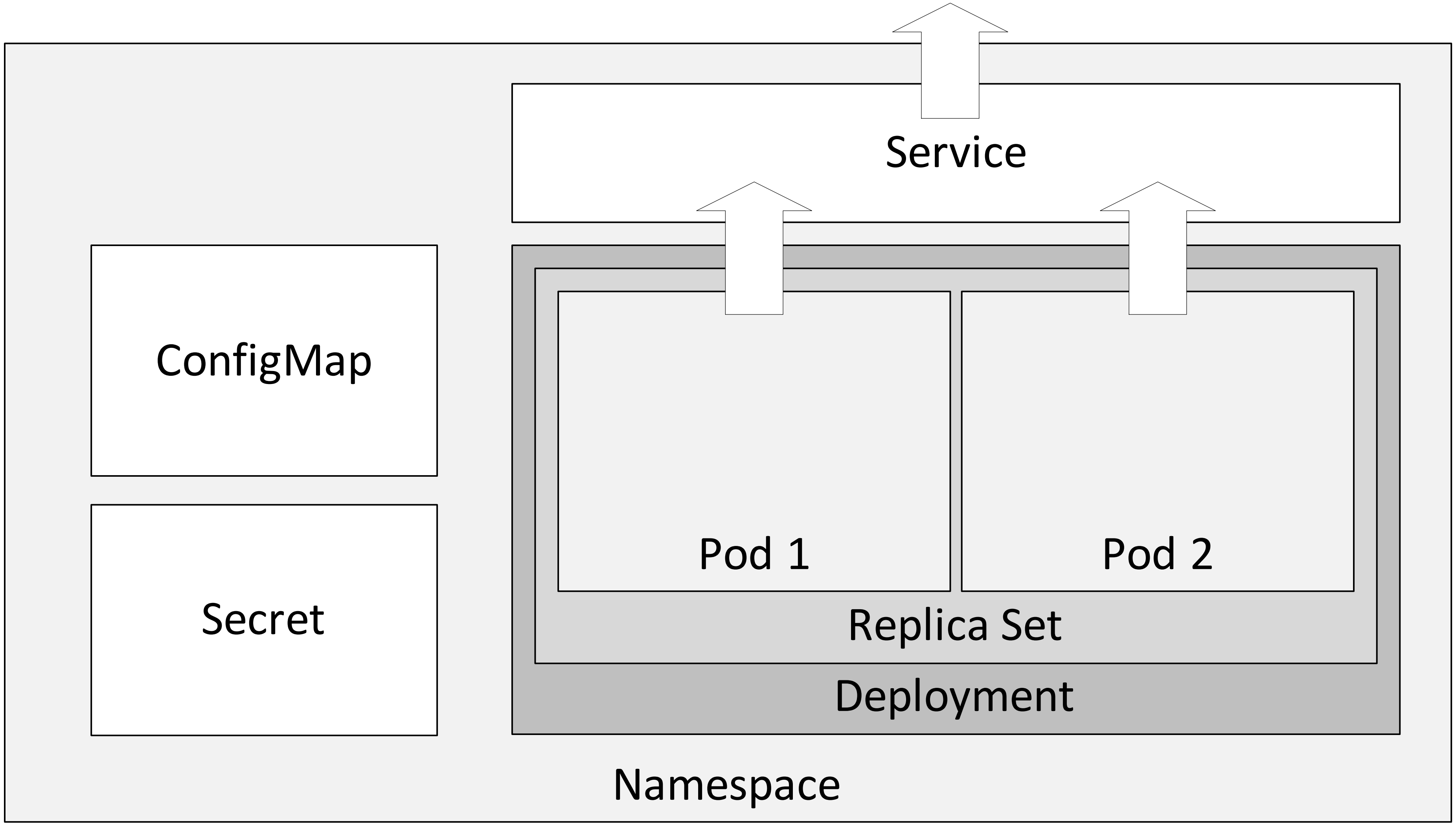
http://jsch.cz/k8slab1

Kubernetes Resources

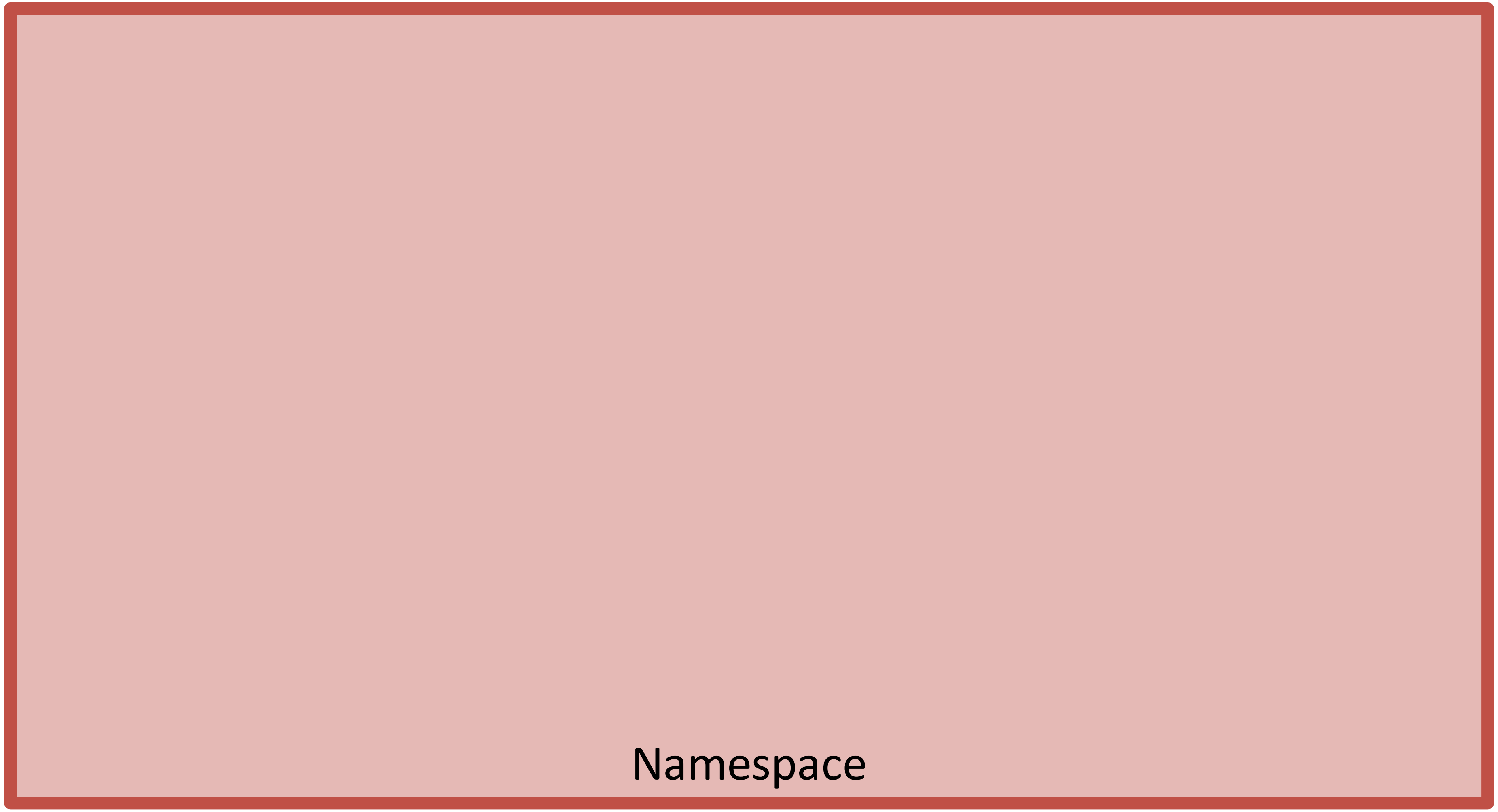
Resources

- Kubernetes API is built out of resources
  - There are many different resource types
  - Each resource has ist own purpose
  - The different resource types work together like Lego bricks
  - To deploy an application, usually several different resources are used

Resources


Service
ConfigMap
Pod 1
Pod 2
Replica Set
Secret
Deployment
Namespace

Namespaces



Namespaces

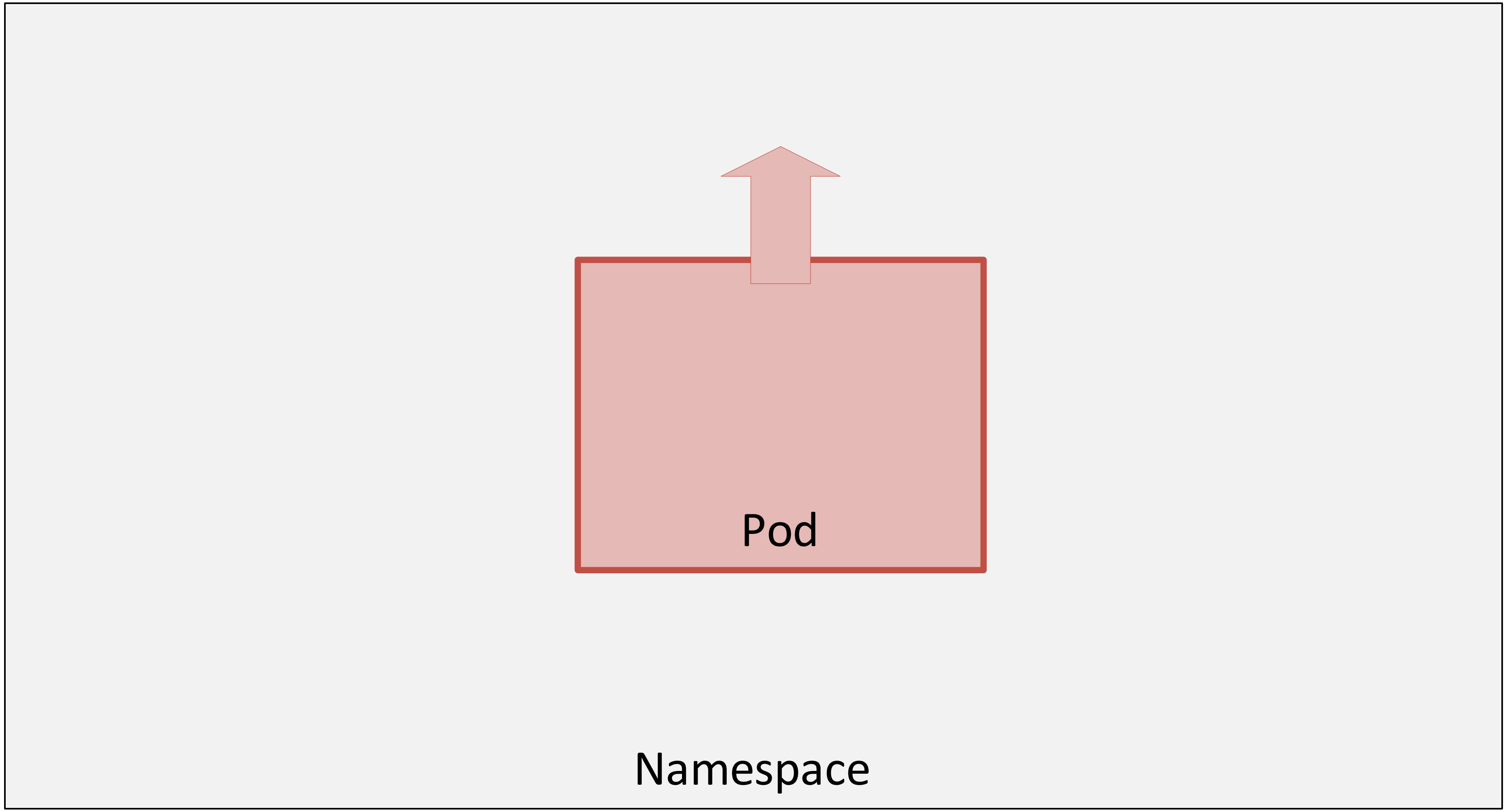
- Most resources are split between namespaces
  - Avoid naming conflicts
  - Isolation between different applications
  - Resource limits
- „*kube-system*“ namespace contains the system components
- „*default*“ namespace is used unless another namespace is specified
- Managed through the API
  - `kubectl get ns`
  - `kubectl create ns schojak`
- Use the „`--namespace=schojak`“ option to list resources from different namespaces

Demo 7: Namespaces

Labels

- All resources in Kubernetes have labels
- Labels are used not only to identify the individual resources, but also to establish links between them and many other tricks

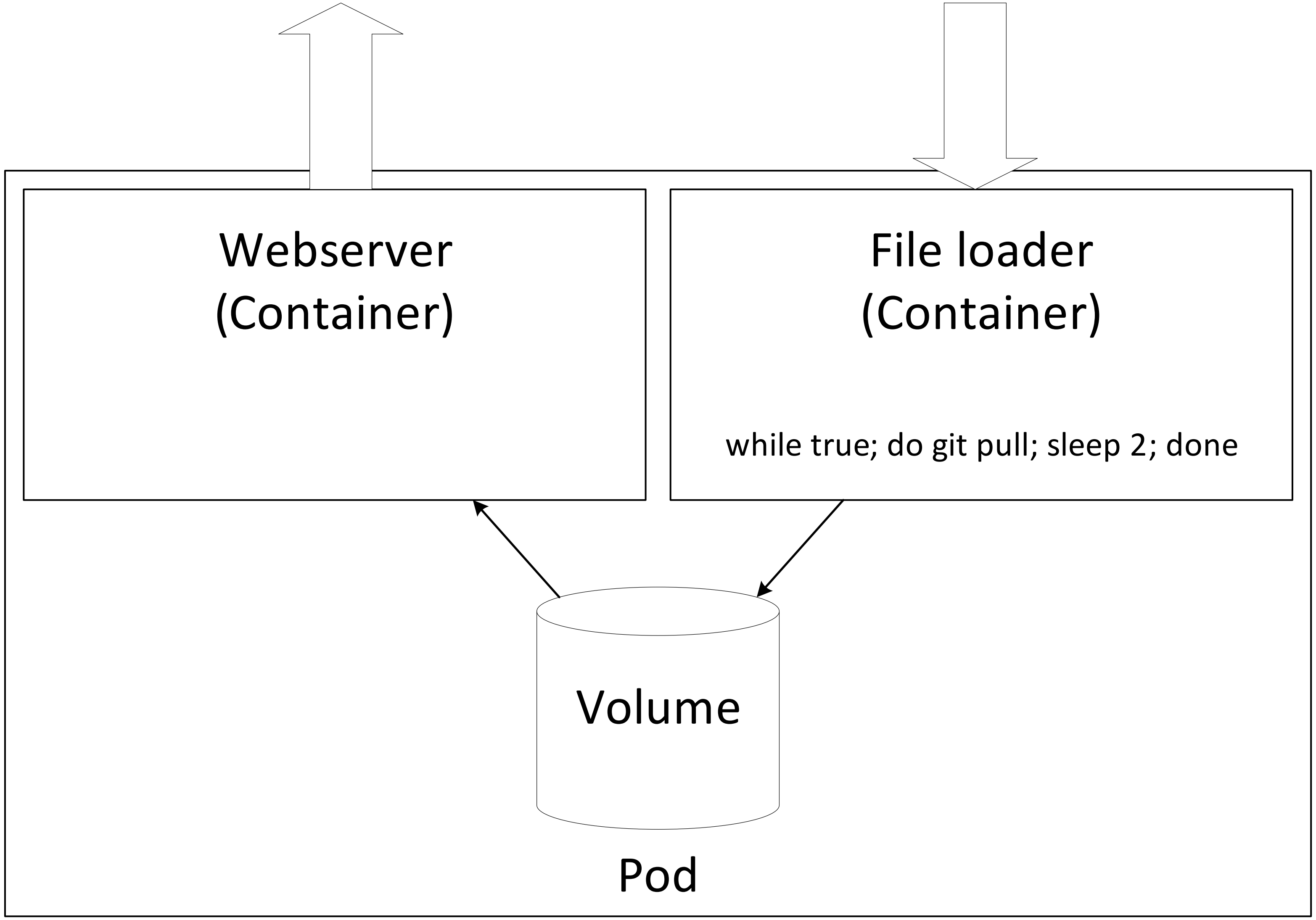
Pods



Pods

- Smallest computing unit in Kubernetes
- Every pod has its own IP address
- Contains one or more containers
  - All containers from one Pod instance are running on the same node
  - Containers in one Pod can easily communicate, share data and resources
  - Containers share Pod‘s IP address and one port space
- Why should you run more container in single pod
  - Tight coupling between applications
  - Logging and monitoring adapters
  - Proxies, bridges and adapters
  - Local cache managers
  - File loaders

Pods


Webserver
(Container)
File loader
(Container)
while true; do git pull; sleep 2; done
Volume
Pod

Pods

- Specified using YAML or JSON
- API version and resource type
- Pod name and labels
- Might specify namespace
- Containers and their ports
- Resource limits
- Volumes

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: redis-Django
  labels:
    app: web
spec:
  containers:
    - name: key-value-store
      image: redis
      ports:
        - containerPort: 6379
    - name: frontend
      image: django
      ports:
        - containerPort: 8000
```

Pods

- Not durable on their own
- When Pod is deleted, applications inside receive TERM signal and have 30 seconds to stop
- Pods are not automatically restarted
- Health and readiness probes:
  - HTTP GET
  - TCP SOCKET
  - Exec

Pods

- Init containers
  - Are run before the main containers are started
  - Can be used to prepare the environment
    - Generate config files
    - Download data
- Resource management
  - Requests and limits
  - CPU and Memory
- Environment variables
  - Used to configure the container(s)

Demo 8: Pods & Labels

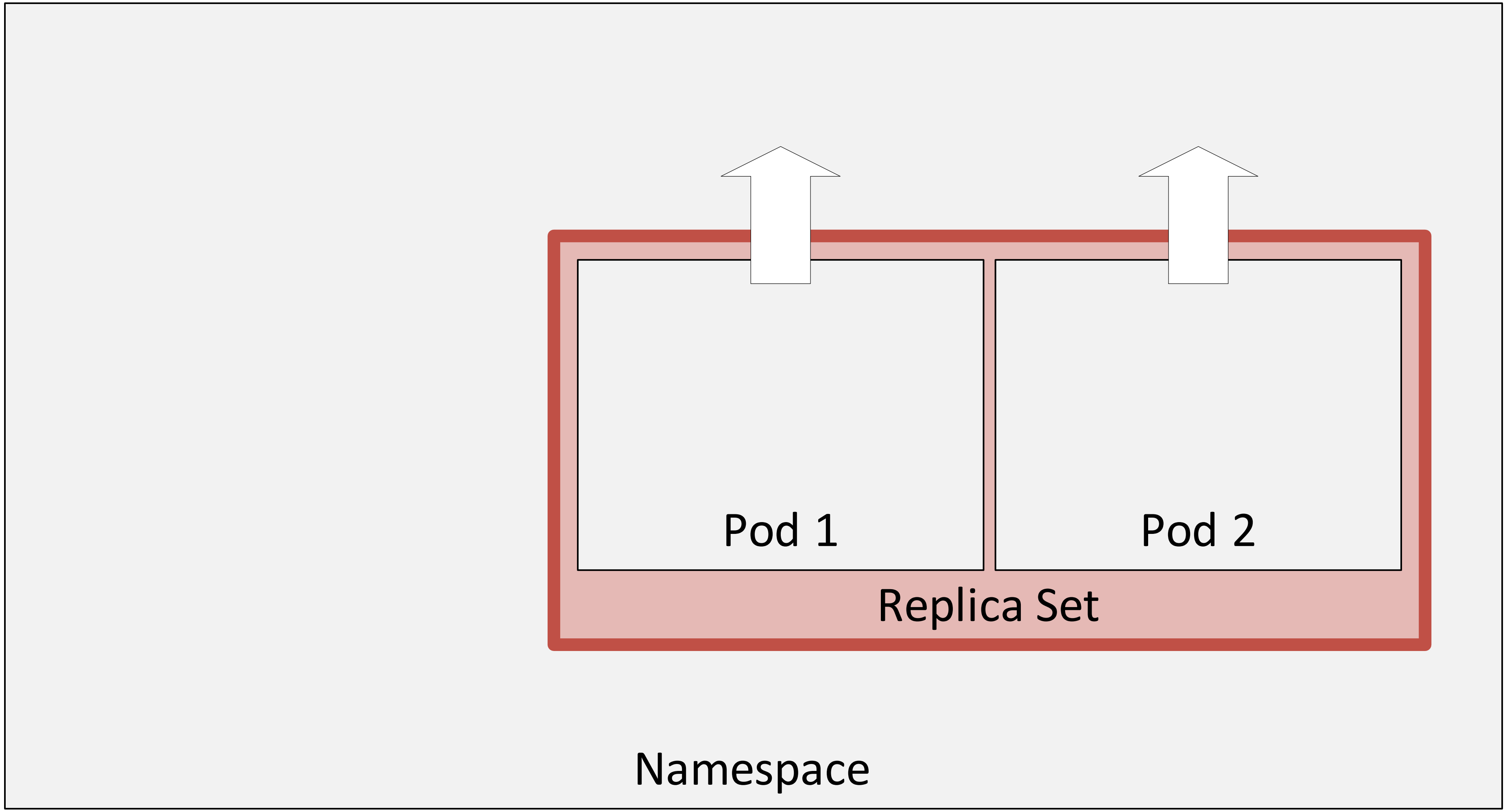
Replication controllers

- Ensures that Pods are running in required number of replicas / instances
  - Starts new Pods if needed
- The RC specification contains a Pod template which is used to create the Pods
- Pods created by RC have no fixed names and cannot be easily addressed directly
- Rolling updates supported only from client side
- Deprecated

Replication controllers

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    app: nginx
  template:
    metadata:
      name: nginx
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

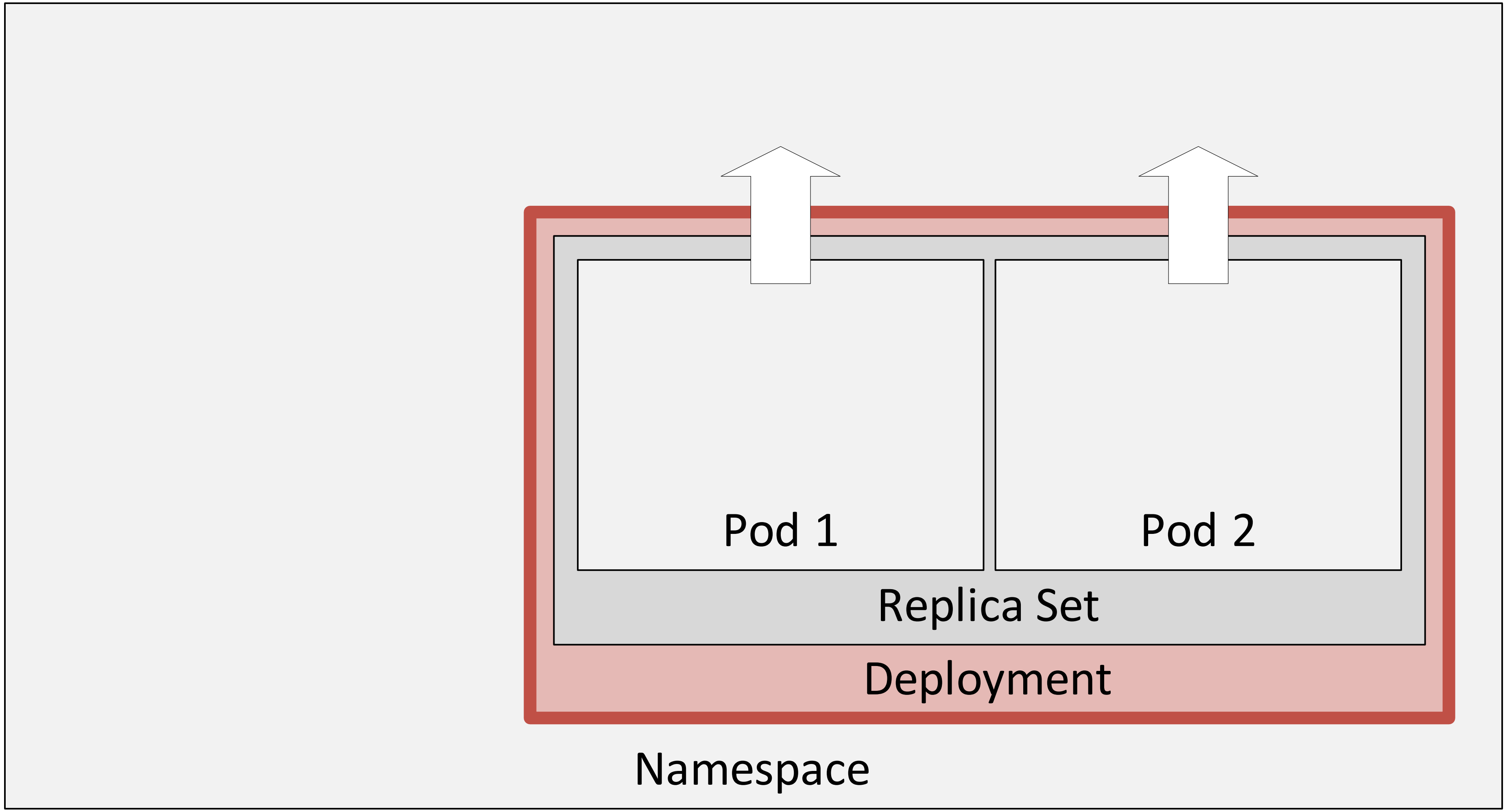
Replica Sets



Replica Sets

- „Next generation“ Replication Controller
- Improved selectors over replication controllers
- Usually used not independently but through Deployments

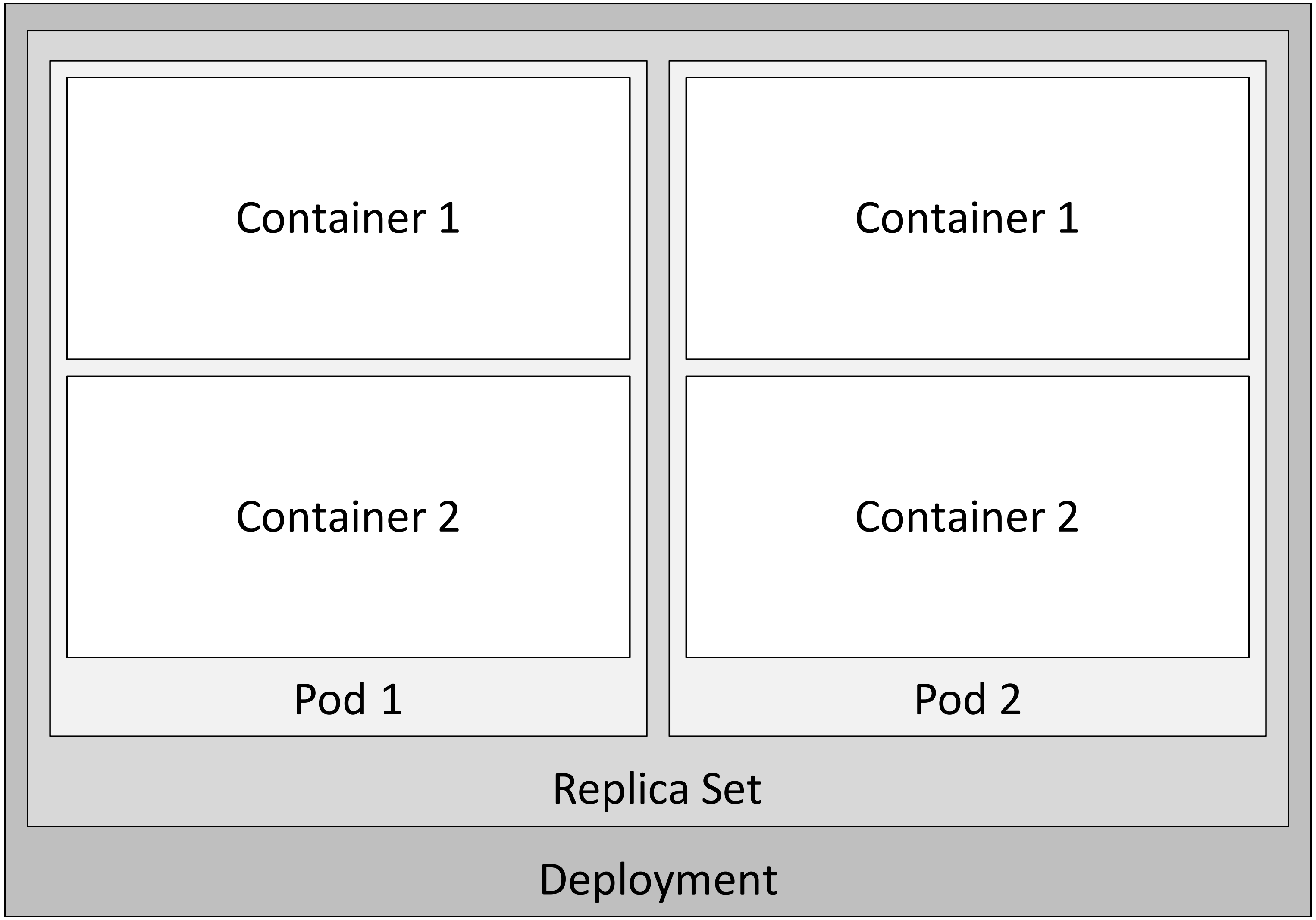
Deployments



Deployments

- Usually used to deploy your application
  - Instead of using Pods, Replica controllers or Replica Sets directly
- Supports server-side rolling updates and canary deployments
- Creates ReplicaSets which create Pods

Deployments

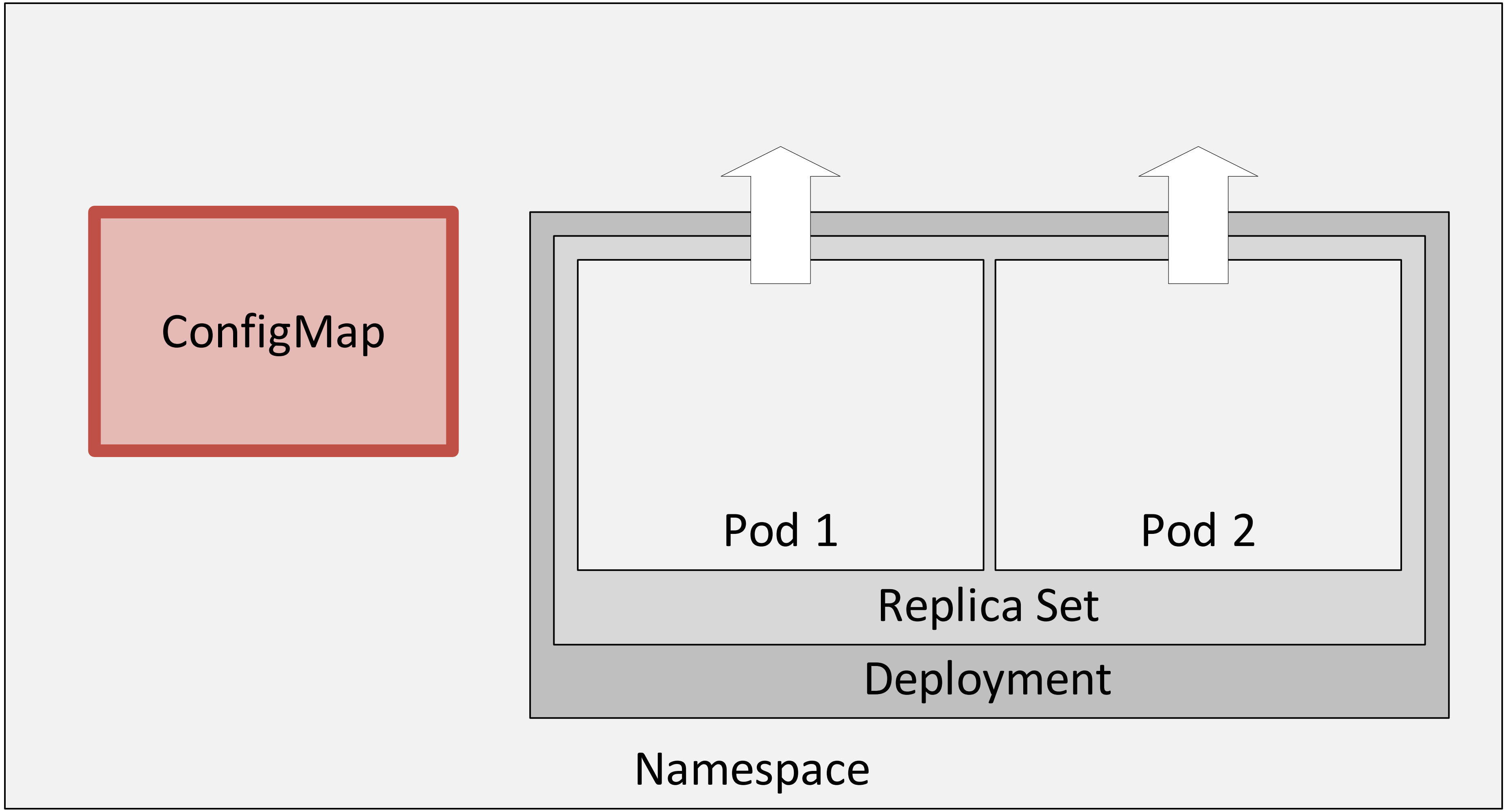


Deployments

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

Demo 9: Deployments

ConfigMaps



ConfigMaps

- Can be created using kubectl
  - From YAML / JSON files
  - Generated from files / directories which should be in the config map

ConfigMaps

- Can be mapped to environment variables

```
env:
  - name: MY_SPECIAL_VARIABLE
    valueFrom:
      configMapKeyRef:
        name: my-config-map
        key: my-key
```

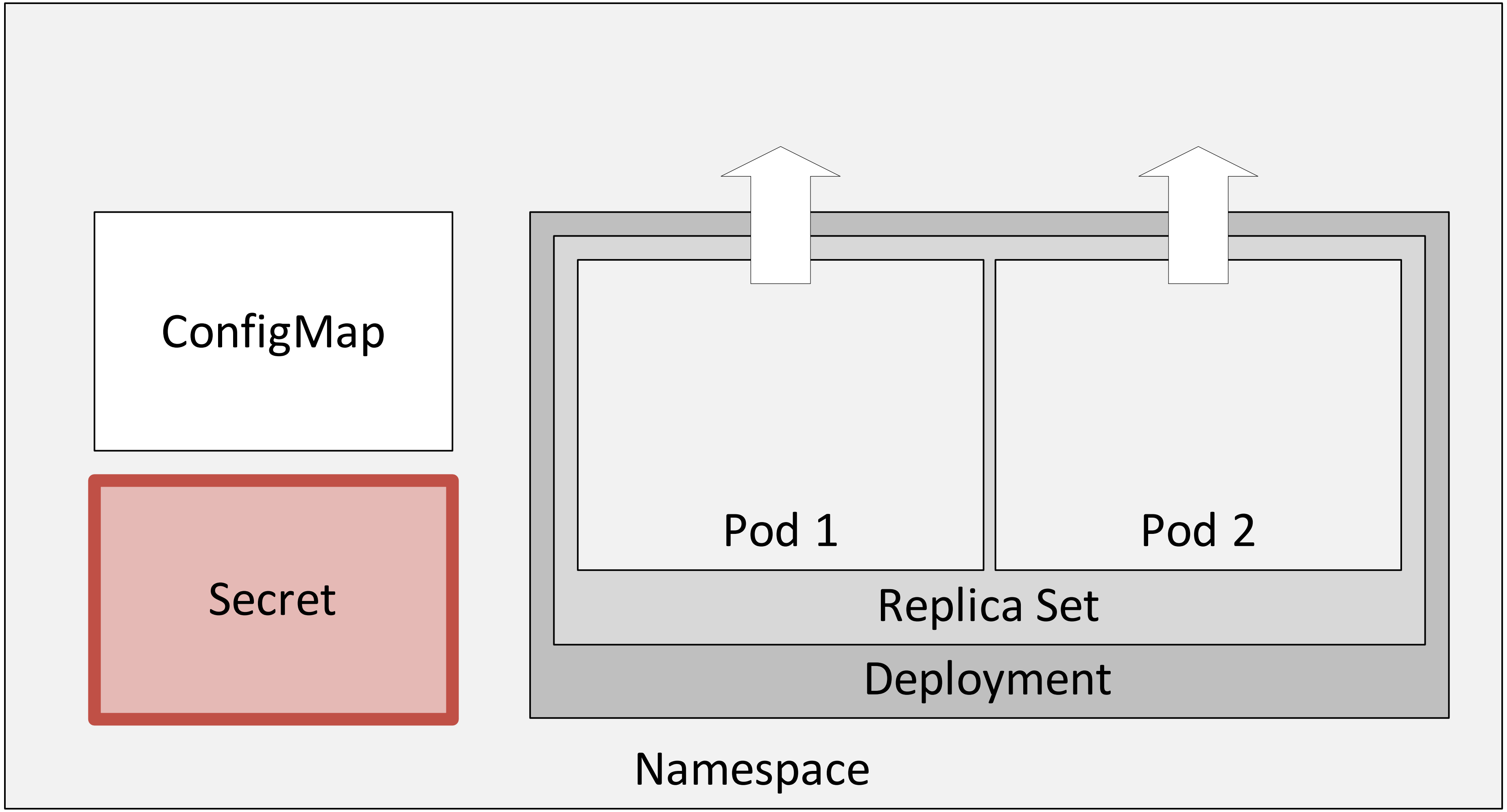
ConfigMaps

- Or mapped to a volume inside the Pod

```
volumeMounts:
      - name: config-volume
        mountPath: /etc/config
…
volumes:
    - name: config-volume
      configMap:
        name: my-config-map
```

Demo 10: ConfigMap

Secrets

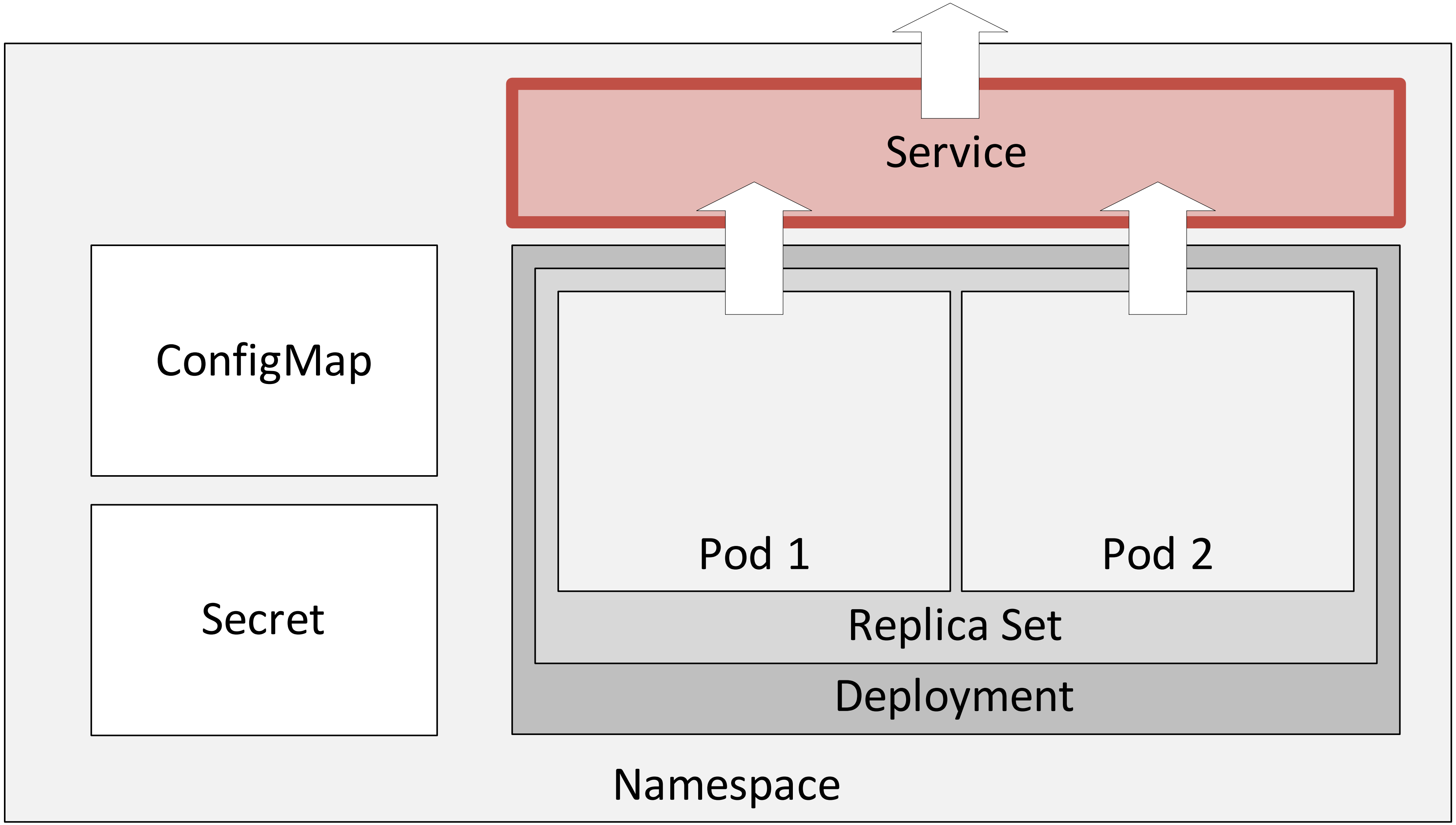

ConfigMap
Secret
Pod 1
Pod 2
Replica Set
Deployment
Namespace

Secrets

- Used to store secrets
  - Passwords
  - Private keys
- Assigned to the node only when some Pod needs them
- Deleted together with the Pod
- Stored in the etcd server
- Creation and usage very similar to config maps

Demo 11: Secrets

Services


Service
ConfigMap
Secret
Pod 1
Pod 2
Replica Set
Deployment
Namespace

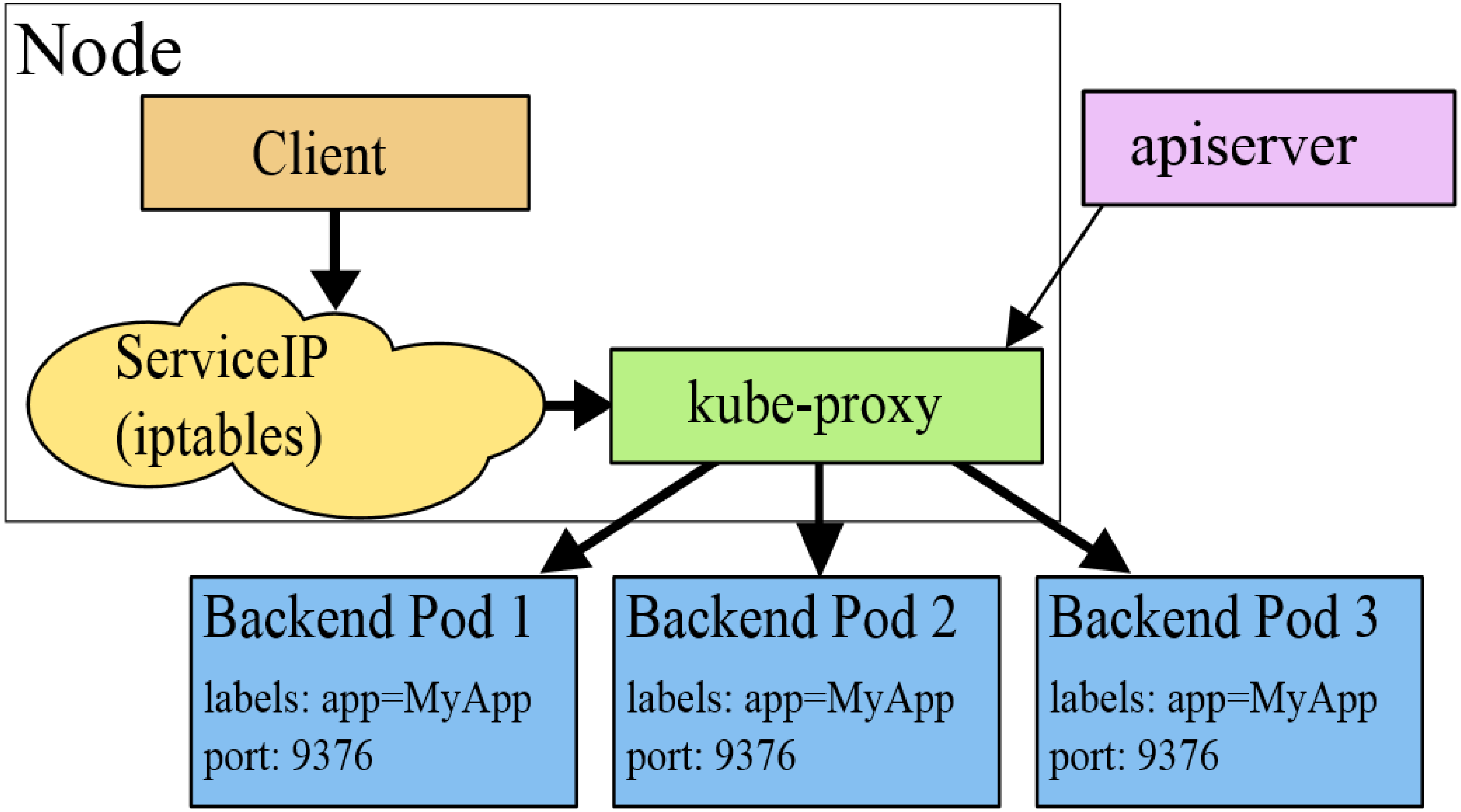
Services

- Containers are running within Pods
- Pods created by RCs / Deployments don‘t have stable names / IP addresses
  - Cannot be addressed from other applications
- Service targets set of one or more pods (based on Label selector)
  - Maps Pod / Container ports to a service ports
  - Exposes the underlying service for discovery
  - Provide stable address using IP address / DNS name
  - DNS names are based on „*service-name.namespace*“ pattern
  - Service hosts and ports are also exposed via environment variables

Services

- Different service types
  - ClusterIP
    - Assigns internal IP address / hostname
    - Accessible only from within the Kubernetes cluster
  - NodePort
    - Binds the service to the Node where the Pod is running
    - Usually used only for special purposes (debugging)
  - LoadBalancer
    - Creates a load balancer which is accessible from outside
    - Work usually on public cloud (AWS, GCP), where it creates its native load balancer

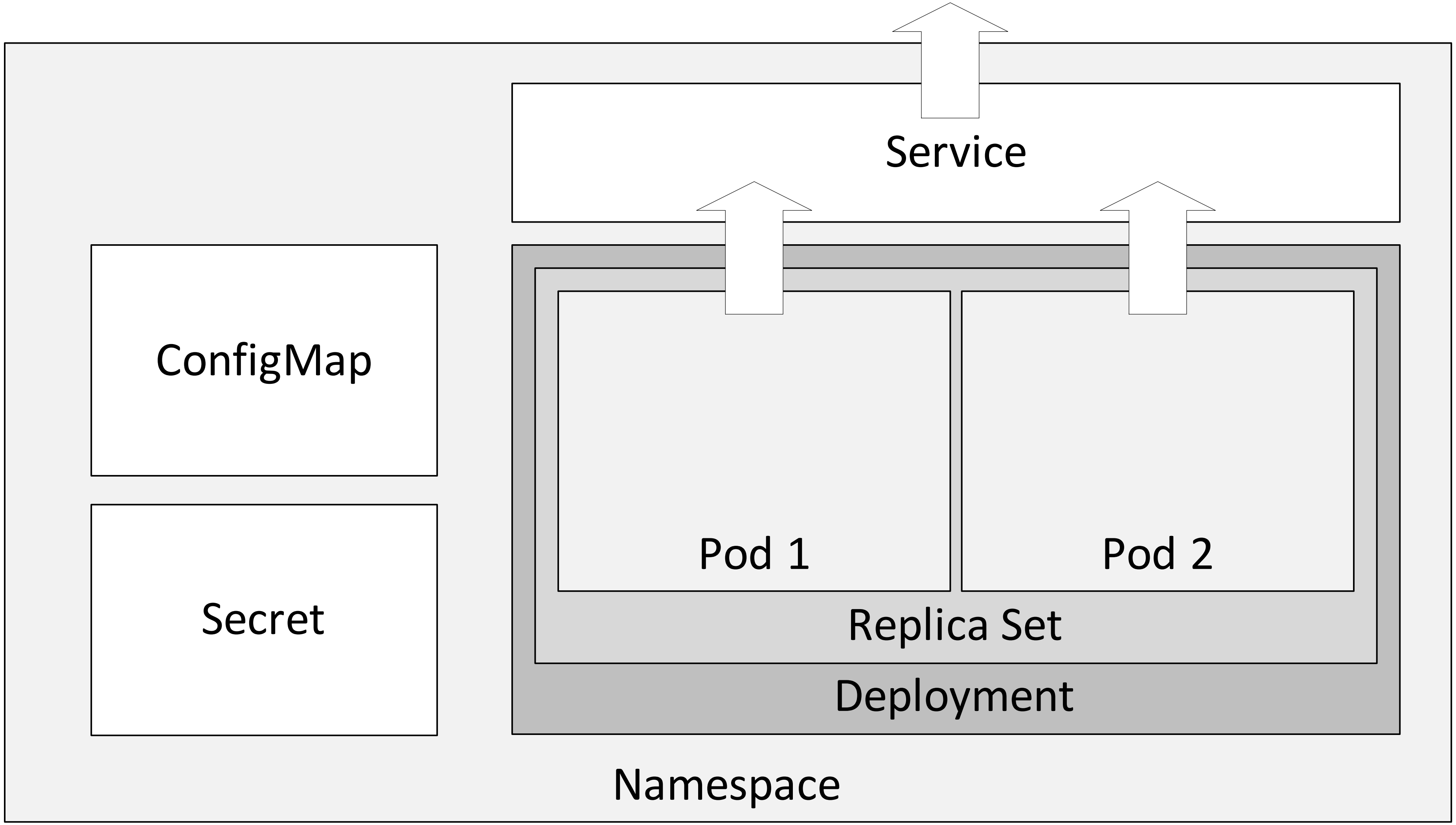
Services



Source: kubernetes.io

Demo 12: Services

Microservices deployment



Lab 2: My first application

http://jsch.cz/k8slab2

Volumes

- Volumes map disks to Pods
- Each volume has name and type

```
volumes:
  - name: my-vol
    emptyDir: {}
```

- Volumes can be mounted into containers using VolumeMounts
  - One volume can be mounted to multiple containers in the same pods
  - Mount paths can be different for each container

```
volumeMounts:
  - mounthPath: /my/volume
    name: my-vol
```

Volumes

- Different volume types:
  - emptyDir
    - Empty directory on the node where the pod runs, which is deleted when the Pod dies
  - hostPath
    - Path on the node where the pod is running, which survives the Pod deletion
  - NFS
  - iSCSI
  - CephFS
  - GlusterFS
  - AWS/GCP disks
  - Persistent Volume Claims
  - etc.

Persistent Volumes

- Abstracts the storage for persisitent disks
  - Used to map Kubernetes volumes to disk as provided by infrastructure (AWS, GCP, Gluster)
  - Thanks to the abstraction, your application doesn`t care where the Kubernetes cluster runs
- Maps against specific physical / virtual disk
  - The persistent volume it self is not portable
  - But all resources referring to it by its name are portable
- Reclaim policy
  - What happens when the disk is returned … should it be deleted or retained?

Persistent Volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mongo-volume
  labels:
    type: local
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  awsElasticBlockStore:
      volumeID: vol-d371e406
      fsType: ext4
  persistentVolumeReclaimPolicy: Delete
```

Storage Class

- Used to dynamically provision Persistent Volumes
  - Specifies the type of the disk (e.g. magnetic, SSD, IOPS)

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: ssd
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
```

Persistent Volume Claims

- Used to claim Persistent Volume
  - Used within a pod
  - Reserves (claims) the persistent volume
- Can be used in combination with StorageClass to dynamically provision volumes
- Claims are referred to by volumes inside the Pods

Persistent Volume Claims

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: mongo-storage
  namespace: database
  annotations:
    volume.beta.kubernetes.io/storage-class: "ssd"
  labels:
    name: mongo-storage
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

Demo 13: Persistent Volumes

Lab 3: Multi-tier application

http://jsch.cz/k8slab3

DaemonSets

- Similar to Replication Controllers
- Run single Pod on every single node of the cluster
- Use cases:
  - Network pods
  - Log collectors
  - Storage daemons
  - Monitoring daemons

Demo 14: DeamonSets

StatefulSets

- Running stateful applications is hard!
- Stateless applications have no need to address replicas individually
  - Treat them as cattle and use deployment
- Stateful applications often need to address replicas
  - Treat them as pets
- Deployments name their pods randomly and make them not addressable
- Alternative to deployment of stateful applications using several separate deployments with single replica are Stateful Sets

StatefulSets

- Named PetSets before Kubernetes 1.5
- Stateful sets treat pods as pets
  - Each pods is created with index instead of random name
  - Index doesn‘t change when you move to different node or restart the pod
  - Thanks to the fixed index, pods have fixed addresses and can be used for clustering

Demo 15: StatefulSets

Jobs

- Provide support for batch jobs
- Job starts a Pod and monitors it
  - If the Pod fails it is restarted
  - If the pod completes the Job completes as well
- Jobs can run Pods in parallel replicas

Jobs

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi
spec:
  template:
    metadata:
      name: pi
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```

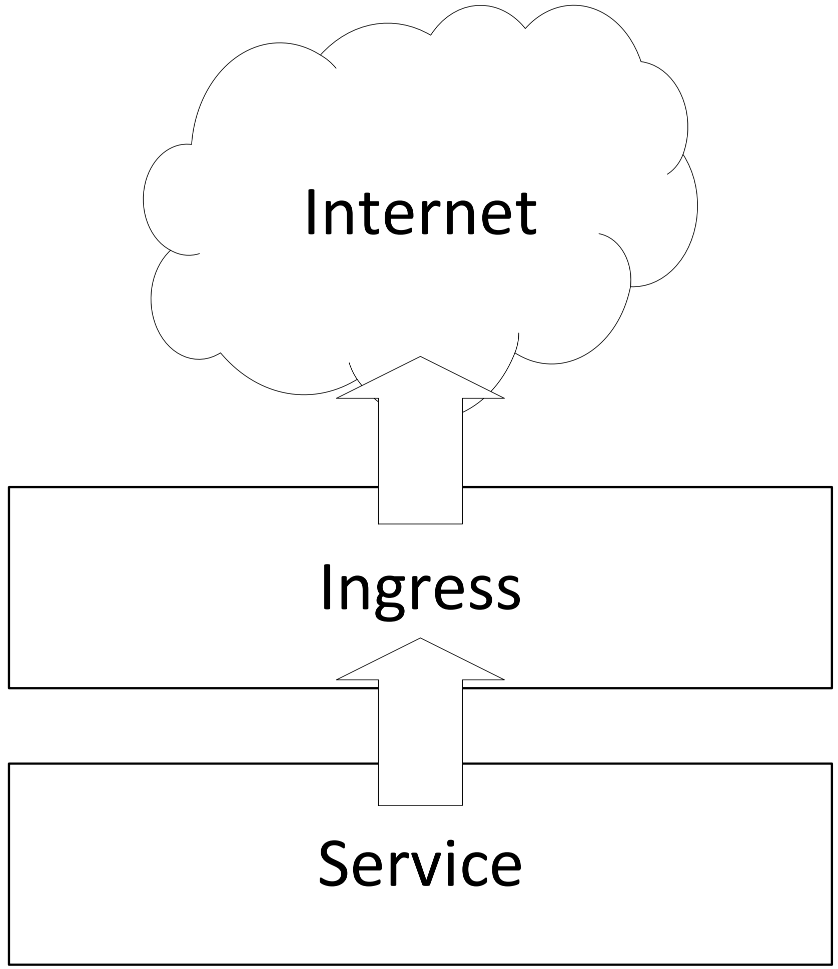
Demo 16: Jobs

Ingress

- Load balancer type service works only on Layer 4 (TCP)
- Ingress stands between the internet and the internal services
- Works as a Layer 7 load balancer
  - Understands HTTP
  - SSL termination
  - Advanced HTTP request routing
- Implemented usually as Nginx proxy

Ingress

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: test
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: s1
          servicePort: 80
      - path: /bar
        backend:
          serviceName: s2
          servicePort: 80
```



Resource Quotas

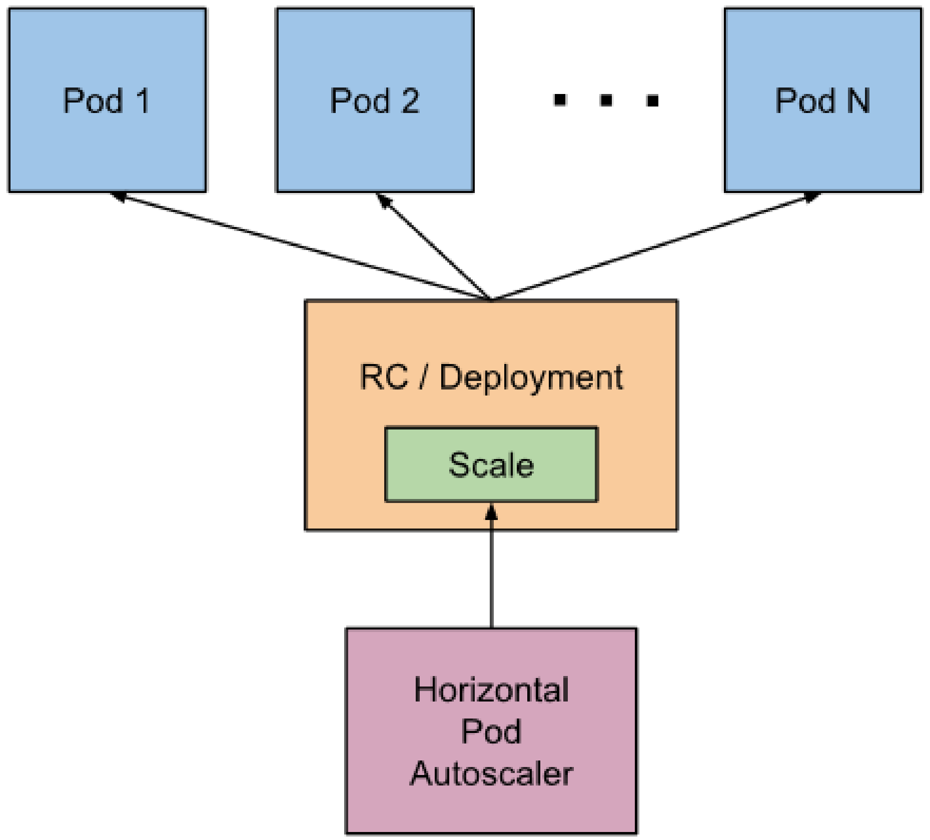
- Applied to Namespace
  - Can limit the number of resources consumed per namespace
  - Bound to different resources
    - CPU
    - RAM
    - Storage volume and number of volume claims
    - Number of created resources (Pods, Services, etc.)
  - Requests vs. Limits

Auto Scaling

- Cloud Native applications should support scaling and elasticity
- Horizontal Pod Auto-scaler can scale pods up and down
  - Plugs into Replication Controller / Deployment
  - Periodically queries the resource consumption and triggers scaling
  - Based on CPU consumption or on custom metrics
- Resource monitoring (Heapster) is needed for working autoscaling

Auto Scaling

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: php-apache
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: extensions/v1beta1
    kind: Deployment
    name: php-apache
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```



Source: kubernetes.io

Demo 17: Auto Scaling

Third Party Resources

- Kubernetes allow to define custom API resources
  - Such resources support CRUD operations as any other Kubernetes resources
- To make them do something, a custom controller handling such resources has to be deployed
    - Monitors the cluster for resource changes
    - Applies the changes when needed
- Example
  - Provisioning of RDS databases through custom resource
  - http://www.devoperandi.com/kubernetes-automation-with-stackstorm-and-thirdpartyresources/

What is missing

What is missing

- Security
  - Network policies
  - User and Service accounts
- Federation
- Helm
  - Kubernetes package manager
  - Allows easy deployment or prepackaged applications