
Kubernetes Intermediate
A workshop by Jetstack
Scanrail1/shutterstock

Schedule

Action-packed and fast-moving

Morning

- Containers and Kubernetes intro
- Getting a K8S cluster on GKE
- ConfigMaps and Secrets
- Auto-scaling (pods and cluster)

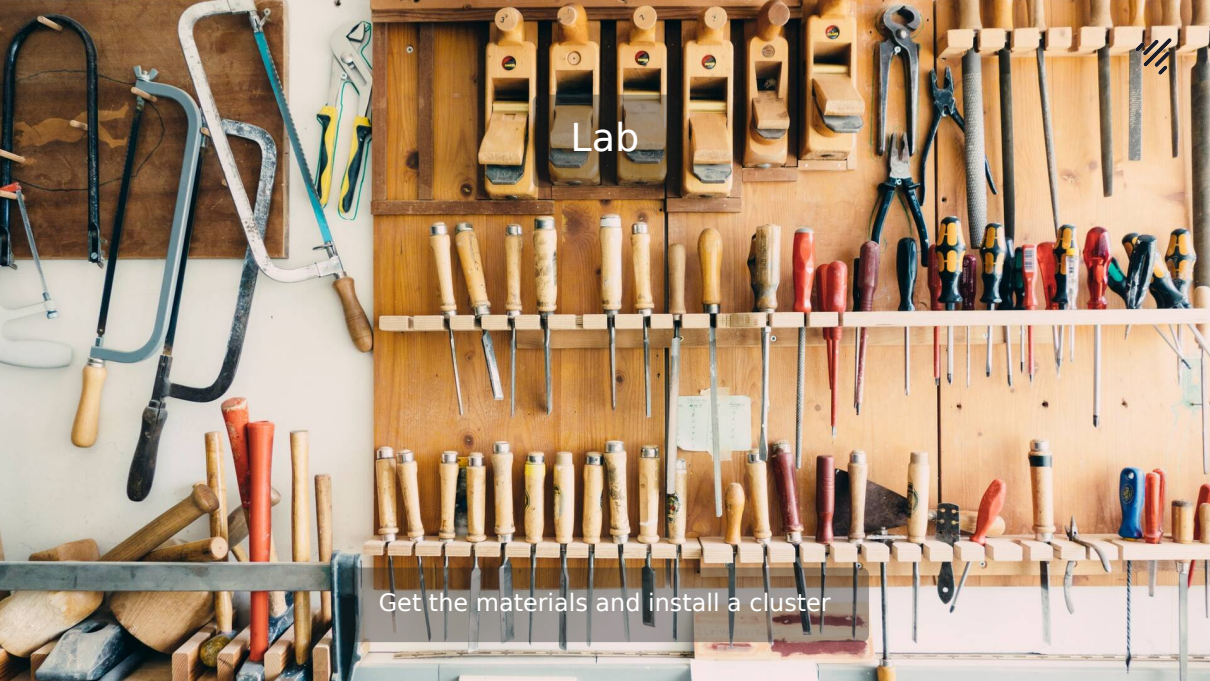
Afternoon

- Control-plane deep dive
- Influencing the scheduler
- Ingress & TLS: SSL for the Sock Shop
- StatefulSet: Deploying some MongoDBs
- Security

Welcome

Each person please say:

- Their name
- What is their background
- How much k8s experience they have
- The main thing they want to learn


Lab
Get the materials and install a cluster

Lab: download the materials

Files and slides for you to keep

http://errge.k8s.school/

- Username: **errge**
- Password: **kubectl2019**
- Download and open the slides in a tab
- Keep this page open, we will need it for the labs
- Download the YAML files and unzip them on your computer
- Send slide and lab feedback to: gergely.risko@jetstack.io
- Join the course chat: https://tlk.io/k8s

Lab: Install tools and create a cluster

Get a 3 node Kubernetes cluster and access it

- GCP (Google Cloud Platform setup)
  - Login to `@gmail.com`, project creation, billing setup, free credits!
  - Install and initialize command line tools on your laptop
  - Enable GKE APIs on GCP
- Create a cluster with the `gcloud` command
  - 3 nodes, that are paid from the $300 we got
  - Master paid and managed by Google for free forever
  - Some sane customizations, feel free to ask for details!
- Connect with `kubectl`
- Add an event to your calendar NOW to delete this GCE project
  - Otherwise the $300 of credits will be spent in 2-3 months
  - You won't be charged even if you forget about this

Introduction

Summary and recap of the beginner course

Intro

Section overview

- Linux containers (cgroups and namespaces) with demos
- Kubernetes goals
- Basic Kubernetes objects:
  - Pod, Replicaset, Deployment
  - Service
  - DaemonSet
  - Storage, StatefulSet
- Readiness and liveness probes
- ConfigMap, Secret and how to update these
- Deploy the sock-shop example into our cluster

Intro

Linux containers = Kernel namespaces + cgroups

- **cgroups** are responsible for resource limits (CPU, memory)
- **namespaces** are responsible for isolation of usually global things:
  - network namespace
  - mount namespace
  - pid namespace (process ids)

Question: which namespace makes it so that a shell in a container can't `kill` a process of the node?

Question: which namespace makes it so that in a container `ps -A` only shows the processes of the container?

Intro

Namespaces demo

- Use `lsns` before and after starting `google-chrome` and/or `chromium`
- Run: `unshare --mount-proc -f -p /bin/bash`
  Then: `ps -A`
- Run: `unshare --mount-proc -f -n -p /bin/bash`
  Then: `ping 1.1.1.1`
  And: `ip link ls`
- Use `nsenter` to join one of the namespaces (use `lsns` to list them)
  Howto: `nsenter -a -t 4399 /bin/sleep 42`

Intro

Namespaces demo

- Run: `unshare -f -p /bin/bash` (so no `--mount-proc`)
  Then: `ps -A`
- Run: `unshare -f -n -p /bin/bash` (so no `--mount-proc`)
  Then: `ping 1.1.1.1`
  And: `ip link ls`

Graduation

- Airgapped container:
  `docker run -it --rm --network=none debian /bin/bash`
- Needs debugging: enter the container's namespaces (except for the network) and install `procps`

Intro

Cgroups summary

- Run the `yes >/dev/null` and the `top` commands
- Use `cat /proc/$$/cgroup` to see in which cgroup we are
- Find it in `/sys/fs/cgroup/cpu`
- Create a child cgroup with `mkdir limit`
- Put the `pidof yes` into the cgroup with `echo 1234 >tasks`
- Look into `cpu.cfs_quota_us` and decrease it drastically!
- Do the same with a bash, how do child processes behave?

Points to note:

- The cgroup kernel API is based on (fake) files
- Processes inherit cgroups, this is why they work like a “container”

Intro

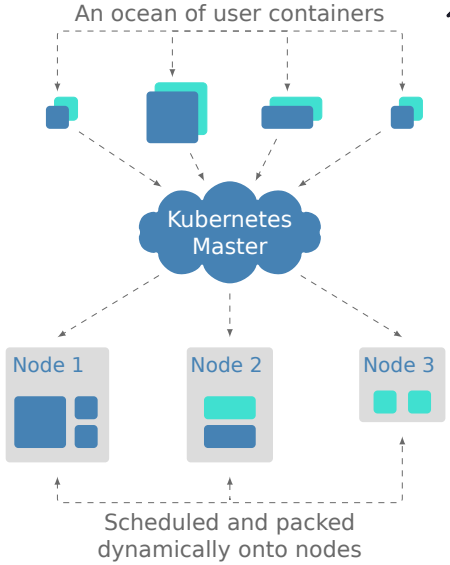
Namespaces and cgroups summary

- Containers are LOT more similar to processes (than to VMs)
- Therefore they are a lot faster to create
- They consume not much of extra resources
- Nothing is safe from the node sysadmin
- Nothing is safe from an attacker who got node root privileges

Kubernetes

Overview

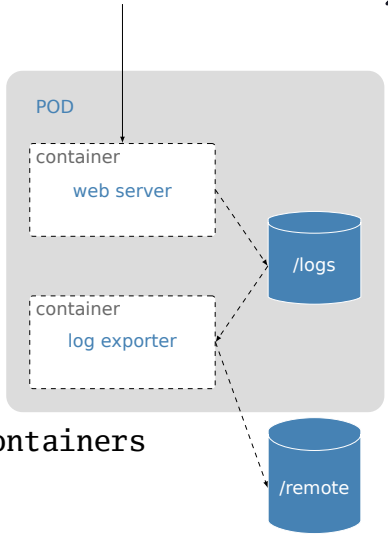
- Handles container “Cattle”
- **Visibility:** Oversight of what is currently running on the cluster
- **Reproducability:** Describe resources in declarative files, infrastructure can be in git
- **Portability:** Works on many different types of infrastructure
- **Resilience and elasticity:** pro-actively monitors, scales, auto-heals and updates



Pod

The basic unit of Kubernetes

- A pod is a collection of containers (one main and zero or more sidecars)
- Some namespaces are separated (mount, UTS, PID)
- Some namespaces are shared (e.g. network and IPC)
- All containers in a pod share fate
- Initialization tasks can be done with `initContainers`



ReplicaSet

Ensure N instances of a pod exist (according to a creation template)

- Often used as a building block for other resources
- Define a pod “template” and it will ensure N replicas exist
- Two replicas don’t have any differentation, they are truly “replicas”
- Does not provide built-in rolling update functionality
- Most end-users should stick to Deployments (next slide)

Deployment

Ideal for “cattle” services

- Deployments manage the lifecycle of ReplicaSets
- Can be used to perform rolling updates of a service
- Two replicas don’t have any differentation, they are truly “replicas”
- We can’t have to be persisted state on the disk inside the container
- Very common resource type

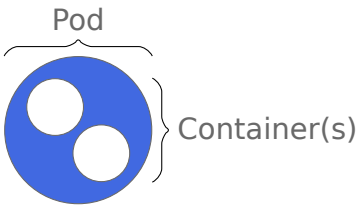
Services

Used to route TCP/UDP (L4) around your cluster

- Provides a basic primitive for load balancing of UDP/TCP
- ClusterIP: default Service type for intra-cluster only traffic
- LoadBalancer: type to expose a port to the Internet
- NodePort: for bare metal (LoadBalancer needs cloud provider)
- Based on labels, not on Deployments (flexibility: canary)
- By default implemented with `iptables` governed by `kube-proxy`
  (Load balancing overhead is distributed over the cluster.)

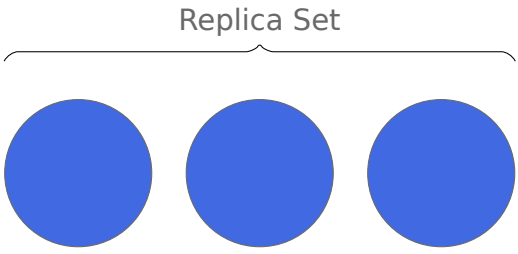
Workloads on Kubernetes

Pods and Containers



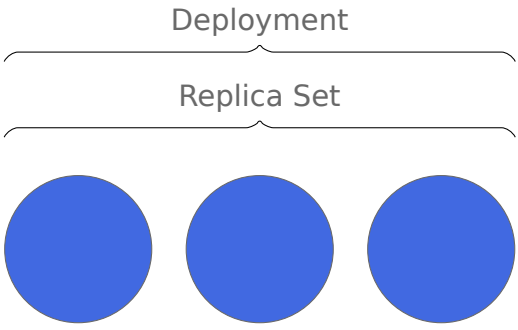
Workloads on Kubernetes

ReplicaSet



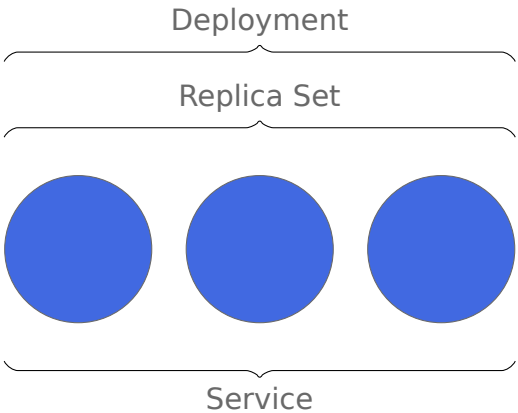
Workloads on Kubernetes

Deployment



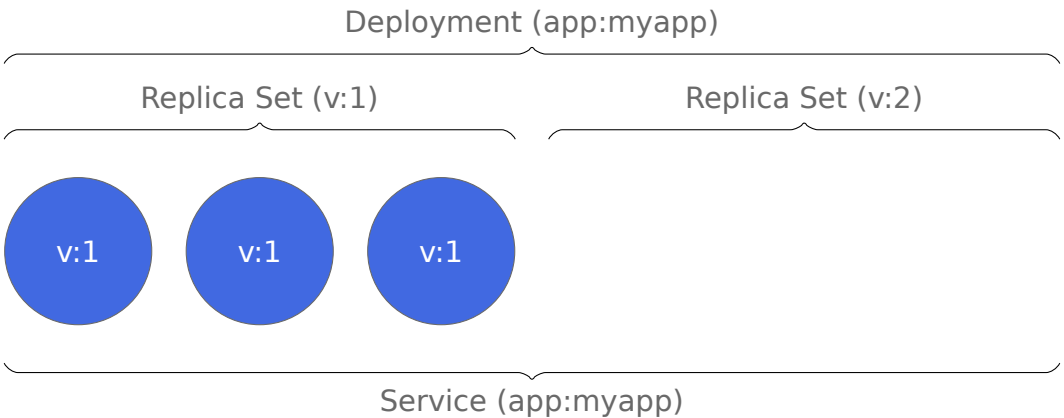
Workloads on Kubernetes

Services



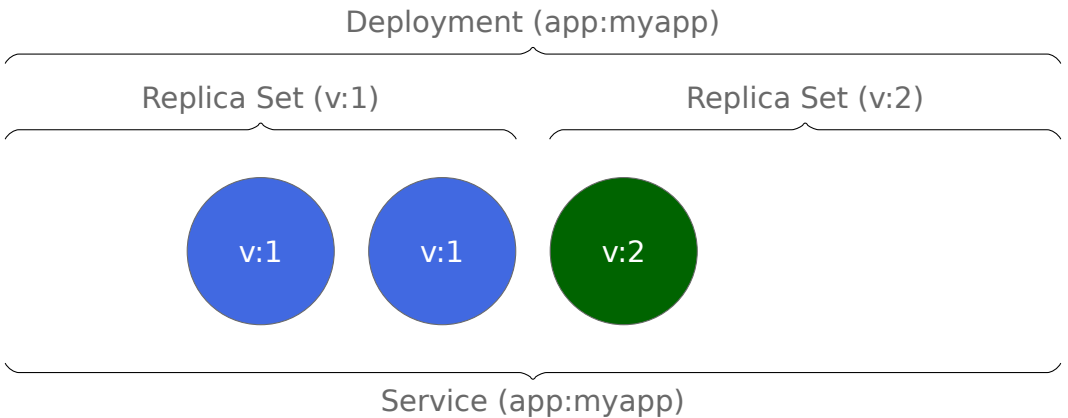
Workloads on Kubernetes

Rolling Update 1



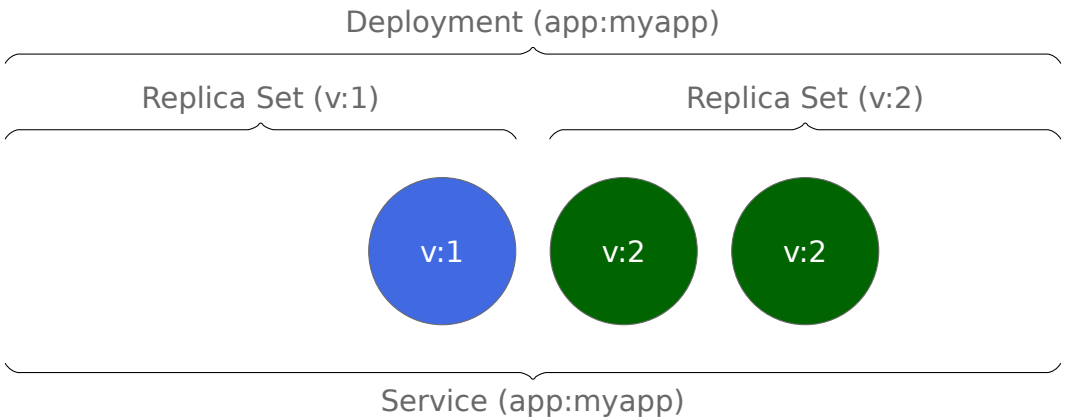
Workloads on Kubernetes

Rolling Update 2



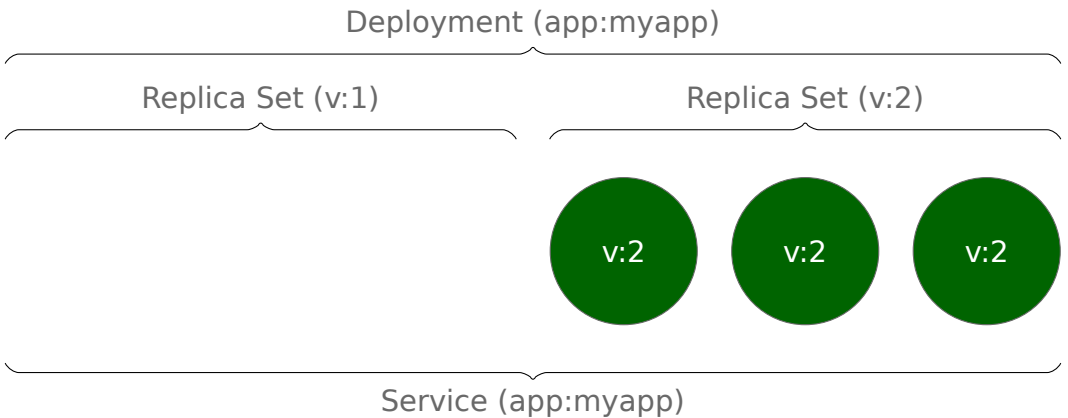
Workloads on Kubernetes

Rolling Update 3



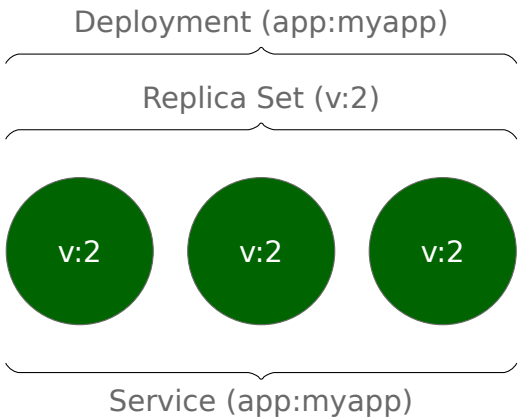
Workloads on Kubernetes

Rolling Update 4



Workloads on Kubernetes

Rolling Update 5



DaemonSet

Run an instance of a pod on each node

- Run exactly one instance of a pod on each node
- Can use `nodeSelector` to target specific types of nodes
- Great for running system services
  - some node monitoring stuff
  - some custom node maintenance
  - `kube-proxy` is run like this in some clusters

Storage: volumes and PV+PVC setups

Kubernetes is only responsible for the wiring

- The cloud providers usually provide some kind of “disk” abstraction
- Bare metal clusters have to setup some open-source solution (ceph)
- Easy but not scalable usage: `volumeMounts` and `volumes` in YAML
- PVC: requesting some disk storage in an abstract way
- StorageClass: serving PVC requests (knows the details of the cloud)
- Used with `StatefulSet` to host “pet” apps on K8S

StatefulSet

Ideal for “pet” services

- Ensure N instances of a pod exist (similar to ReplicaSet)
- Each pod has an “identity” (oridnal), e.g. `pod-0`, `pod-1`, `pod-2`
- Stable hostname
- Persistent storage *per pod*
- Ideal for distributed systems/clustered databases
- In case of master-slave, 0 is usually the master
- Most useful if there is a `StorageClass` for automatic provisioning of `PersistentVolumes` based on `PersistentVolumeClaims` (GKE has this)

Readiness and Liveness Probes

When to serve, when to restart

- Liveness: should a container be restarted already?
- Readiness: should the service route requests to a pod?

ConfigMap

Decouple application configuration from your images

- A K8S object containing some key-value configuration data
- Can be mounted into a container as a volume
- Can be used as environment variables (or command line flags)

ConfigMap Update Guidelines

How to update our ConfigMaps? Do not!

- Volumes + autoreload: crash on syntax error, service unavailable
- Environment variables (or command line flags) read on startup:
  Nothing happens because of a ConfigMap update
  The deployment hash only depends on the deployment
- Okay, then I will use fake labels to enforce a redeploy:
  Popular solution, but problematic to revert

The rule of ConfigMaps is to **never update ConfigMaps**, we only create new ones and then reference them from the Deployments.

Secret

For storing sensitive data

- Behaves the same as a ConfigMap
- But handled a little bit more securely:
  e.g. mounted on a ramdisk on the node when used as a volume

Beware: any Kubelet (and therefore any root user on any node)
can read any Secret in the whole cluster.

Getting ready

Goodies

Here are some goodies!

- https://kubernetes.io/docs/reference/kubectl/cheatsheet/
- Get exact documentation in the command line: `kubectl explain`
- For bash users: `source <(kubectl completion bash)` in any terminal window or in your `.bashrc` or `.profile`.

Getting ready

kubectl versions

- Check: `$ kubectl version` (current client and cluster version)
- Quite usual versioning: major.minor.patch
- Server/client compatibility rule: minor version +-1 difference
- Best to have the same version, if you can
- Symptoms: random commands fail with cryptic messages

Getting ready

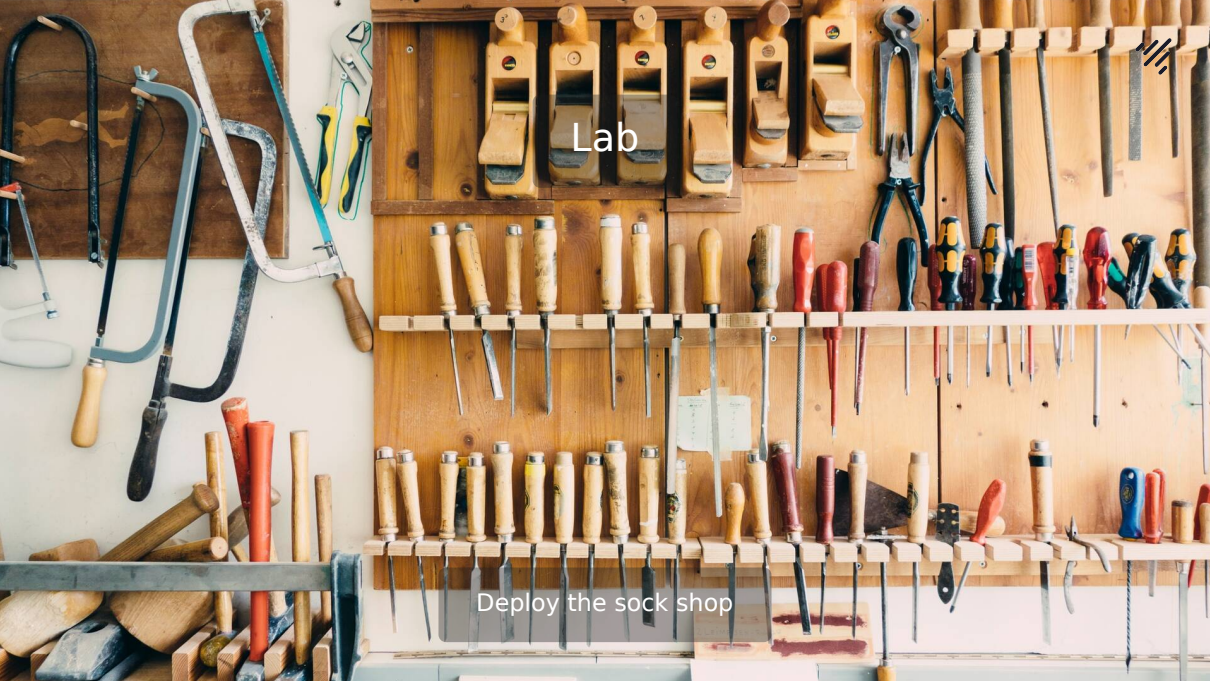
Obtaining specific kubectl versions

How to have the same version?

- OSX: https://storage.googleapis.com/kubernetes-release/release/v1.11.10/bin/darwin/amd64/kubectl
- Win: https://storage.googleapis.com/kubernetes-release/release/v1.11.10/bin/windows/amd64/kubectl.exe
- Lnx: https://storage.googleapis.com/kubernetes-release/release/v1.11.10/bin/linux/amd64/kubectl

Trick on linux:

```
gcloud compute scp <node>:/home/kubernetes/bin/kubectl ~/bin
```


Lab
Deploy the sock shop

Lab: Deploy sock shop

Deploy the entire stack using `kubectl`

- create the namespace with `$ kubectl apply -f`
- create the services with `$ kubectl apply -f`
- create the deployments with `$ kubectl apply -f`
- view the site from your browser
- no cleanup, we will use this sockshop for all the following labs

Autoscaling

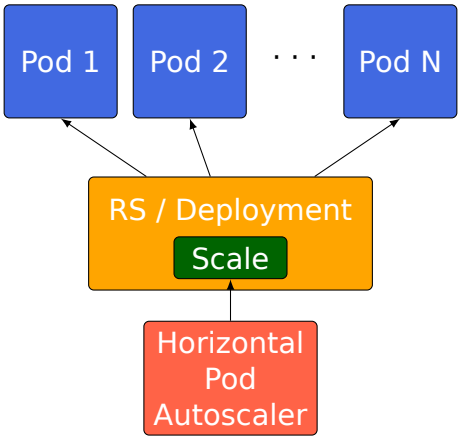
Scaling our apps and clusters

Maximumm/shutterstock.com

App autoscaling

Allow **applications** to scale

- `HorizontalPodAutoscaler` resource
- `v1` supports CPU based scaling
- `v2beta1` supports custom metrics
- Avoids re-implementing scaling logic
- Orthogonal resource that can scale:
  - ReplicationController
  - ReplicaSet
  - Deployment
  - StatefulSet (starting 1.13)
- By default, checks CPU usage every 30 seconds in a control-loop.



App autoscaling

Example

- Horizontal: the same Pod template is used, with same resource settings
- Pod: simply starts more pods, by changing the replica count
- also scales down automatically
- Percentage: of what? Why?

```yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: my-autoscaler
spec:
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 60
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
```

App autoscaling

v2beta1 custom metrics

- CPU scaling can be problematic:
  startup load, periodic garbage collection, query of death bugs
- Better metrics:
  query per seconds, items processed per second, etc.
- These metrics are harder to use: needs loadtesting, needs changing if moving between infrastructures, needs Kubernetes support
- On the Kubernetes side `v2beta1` allows custom metrics
- We can use these to teach K8S about our application
- We will use this new syntax in the Lab
- https://github.com/kubernetes/community/blob/master/contributors/design-proposals/instrumentation/resource-metrics-api.md

App autoscaling

v2beta1 example

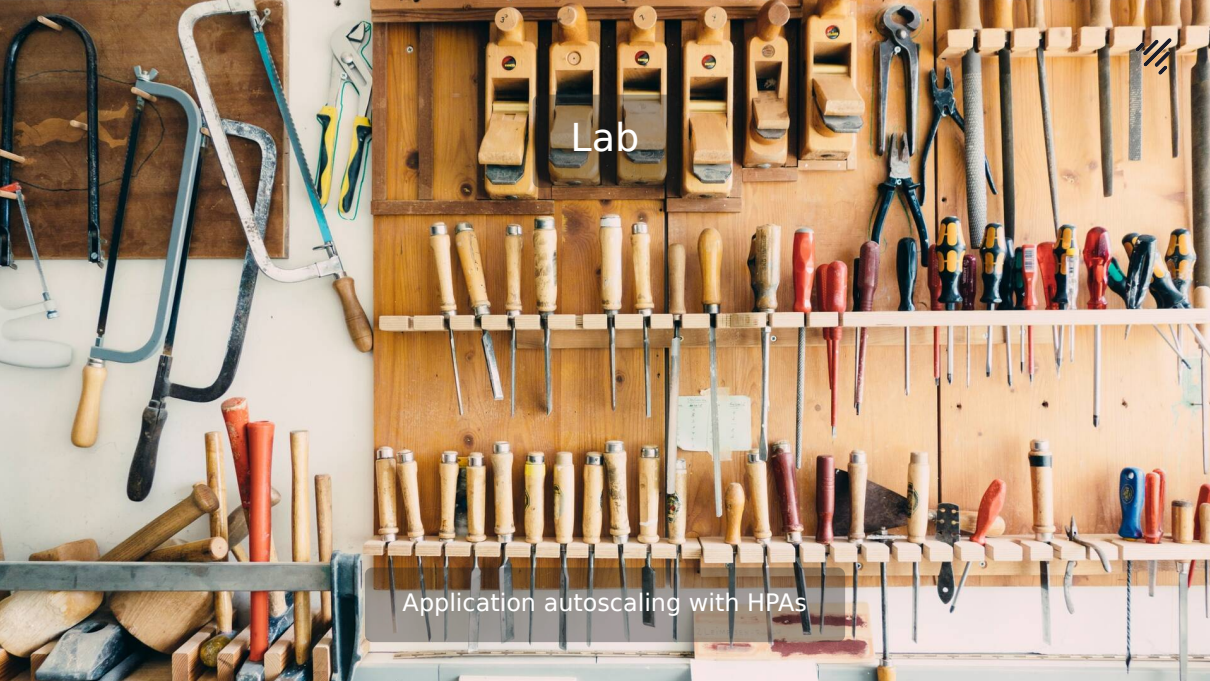
- The v2beta1/v2beta2 supports a lot more
- Some flags can only be changed on the master (no way on GKE)
- v1 is deprecated, but works
- Debugging HPAs: `kubectl describe`

```yaml
apiVersion: autoscaling/v2beta1
kind: HorizontalPodAutoscaler
metadata: { name: my-autoscaler }
spec:
  minReplicas: 2
  maxReplicas: 5
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
  metrics:
    - type: Resource
      resource:
        name: cpu
        targetAverageUtilization: 60
    - type: Pods
      pods:
        metric: { name: request-per-second }
        targetAverageValue: 2k
```

App autoscaling

Reference docs

- https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
- https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/
- if you use the CPU utilization percentage, then it is the percentage of the **REQUEST**
- therefore you have to have requests defined in your pod templates for your deployments


Lab
Application autoscaling with HPAs

Lab: application autoscaling with HPAs

Create an HPA and test it

- Create an HPA resource for the front-end deployment
- Start generating (artificial) load
- Observe scale up
- Stop generating load
- Observe scale down
- Cleanup: delete the HPA

Cluster autoscaling

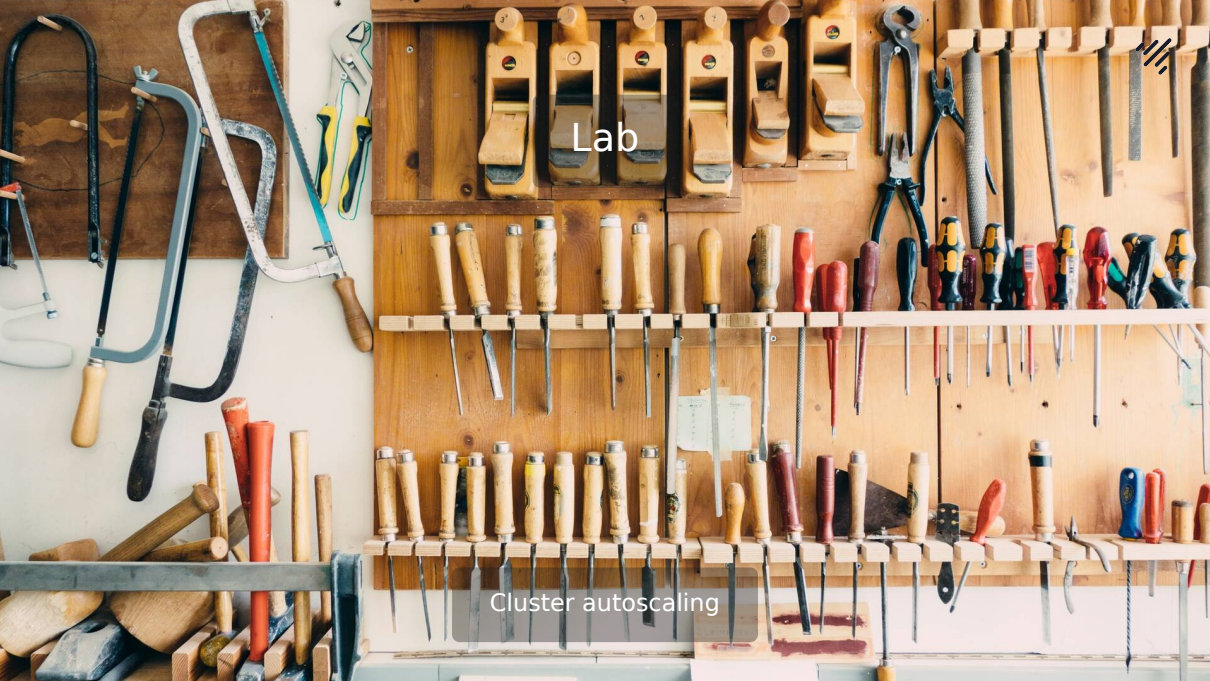
Allow the cluster to scale

- Watches cluster for unschedulable pods
- Triggers scale up or down in response
- Won’t scale on utilization (CPU percentage), only schedulability
- Supports GCP, GKE, AWS and Azure right now
- https://github.com/kubernetes/autoscaler

Cluster autoscaling

Allow the cluster to scale

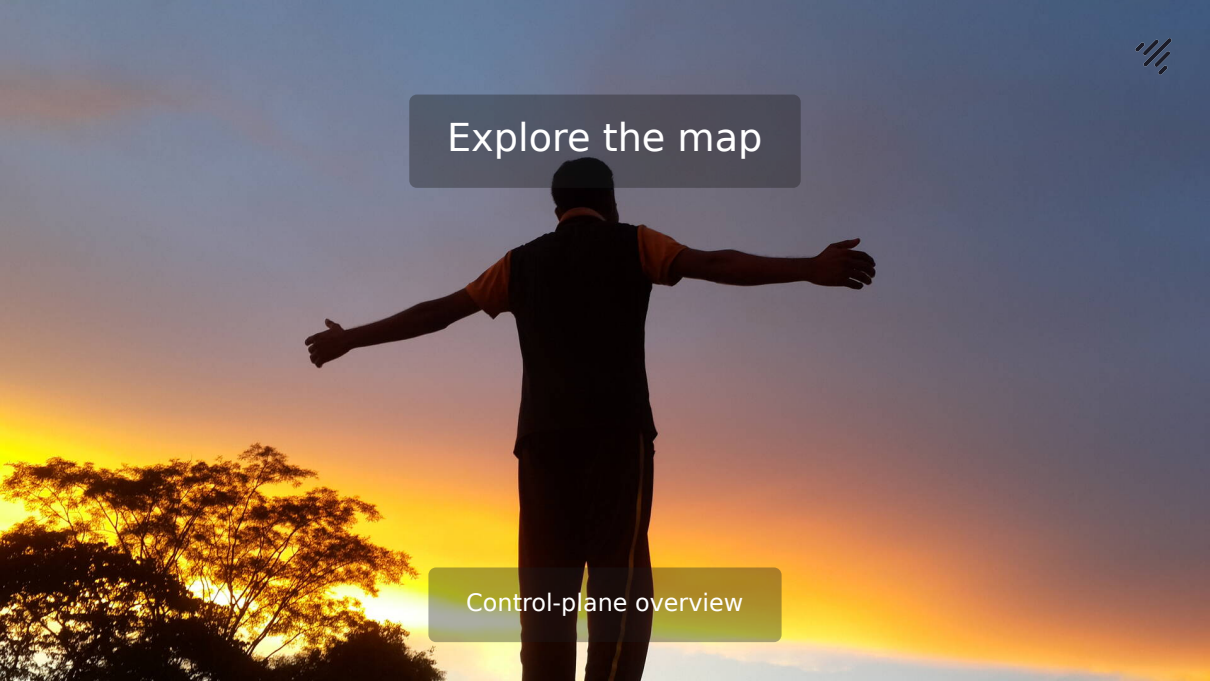
- Not immediately responsive, it can take 5 mins to scale up
- Even longer for scaling down
- Won’t scale down after scale up for 10 mins
- Cannot scale vertically, only horizontally (no new machine types)
- Discuss why CPU-based cloud provider provided autoscalers would be a bad match for K8S
- Solution: set proper requests, HPAs and cluster autoscaler
- We use this in Jetstack internally for Gitlab CI, works well
- There is even a complicated way to set up overprovisioning


Lab
Cluster autoscaling

Lab: cluster autoscaling

Play with the cluster autoscaler

- Set up the cluster autoscaler
- Schedule a lot of pods
- Wait for the scaling to happen
- Remove the pods, wait for the scale down


Explore the map
Control-plane overview

Explore the map

Section overview

We discuss Kubernetes internals here! Keep awake...

These are important to know:

- gives you a chance when you have to debug something
- makes Kubernetes less magical
- makes it possible for you to guess what is possible and what is not
- the components are simple and loosely coupled
- but the consequences can be complicated

Explore the map

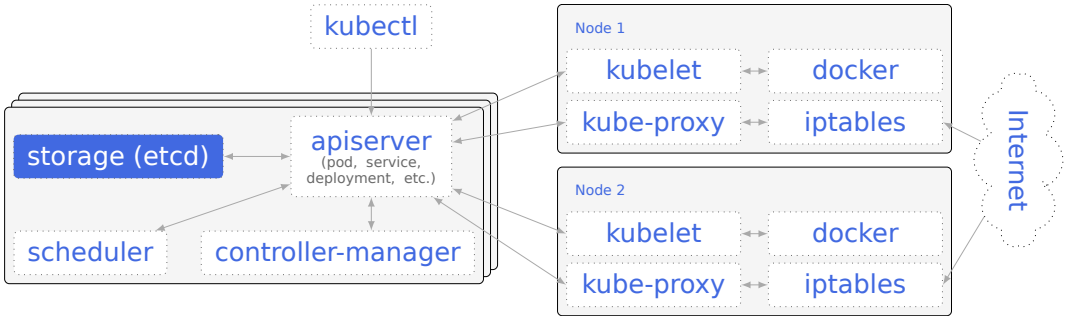
What do we have in an empty cluster?

- Client (how the owner interacts with the cluster)
  - **kubectl**: CLI client for the api server
- Control plane on the Masters (cluster orchestration processes)
  - **etcd**: storage for the state of the cluster
  - **apiserver**: REST API for `kubectl` (uninteresting, parts auto-generated)
  - **scheduler**: schedule unscheduled pods onto nodes
  - **controller-manager**: control loops for cluster operations (e.g. synchronizing desired and actual state, pinging nodes, etc.)
- Nodes (node maintenance processes that run on each node)
  - **kubelet**: talks to the apiserver on the master and to docker locally
  - **docker**: container engine that actually handles the containers
  - **kube-proxy**: implements services by editing `iptables`

https://kubernetes.io/docs/concepts/overview/components/

Explore the map

Overview



Explore the map

kubectl command families

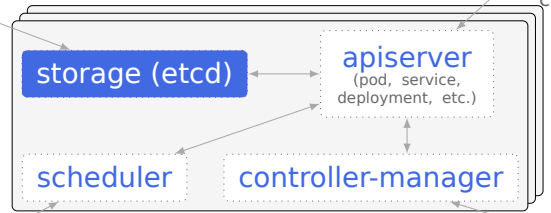
- `kubectl {get,describe} {pod,node,svc,deployments,events,all}`
  Inspect the resources on your cluster
- `kubectl apply -f {pods,nodes,services,deployments}.yaml`
  Create and update resource on your cluster
- `kubectl delete {pods,nodes,services,deployments}`
  Delete resource on your cluster by name
- `kubectl delete -f {pods,nodes,services,deployments}.yaml`
  Delete resource on your cluster by YAML file
- `kubectl rollout {history,pause,resume,undo,status}`
  Handle rollouts for deployments, daemonsets and statefulsets
- `kubectl {explain,logs,label,config,drain,exec,autoscale,...}`
  https://kubernetes.io/docs/user-guide/kubectl/

Explore the map

Control plane: Kubernetes Master

Distributed key/value store using the RAFT consensus algorithm, the **state** of the cluster.

REST API for `kubectl` and other clients, the **interface** of the cluster. (Multi master)



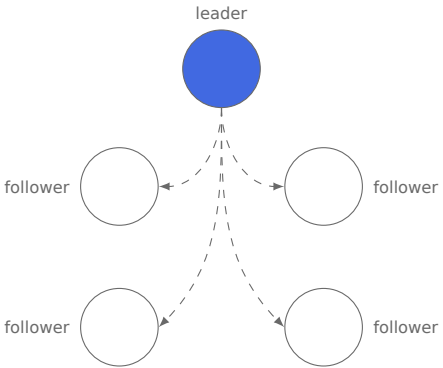
Allocates pods onto nodes based on current resources and constraints, the **resource manager** in the cluster. (Leader-follower)

The control loops for the system. Continuously reads cluster state and triggers actions in response to events, the **brain** of the cluster. (Leader-follower)

Explore the map

Control plane: etcd

- Distributed HA config database
  http://etcd.io/
- Stores small amount of data, but with high availability and consistency
- Based on the RAFT algorithm
- Google origins: Chubby lock service
- thesecretlivesofdata.com/raft/



Explore the map

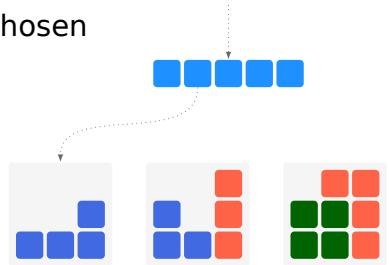
Control plane: apiserver

- A REST server for managing the cluster (used by `kubectl`)
- Control the K8S API from any programming language
- Pluggable authentication: X509, username/password, OAUTH, etc.
- Authorization: RBAC (role-based, roles and role bindings as usual)
- Admission controllers: optional additional policy enforcement (e.g. `AlwaysPullImages` and `PodSecurityPolicy`)
- REST, so uses HTTP verbs: `GET`, `POST`, `PUT`, `DELETE`
- Versioned: currently `/api/v1/{pods,services,deployments,...}`
- kubernetes.io/docs/reference/generated/kubernetes-api/v1.11/

Explore the map

Control plane: scheduler

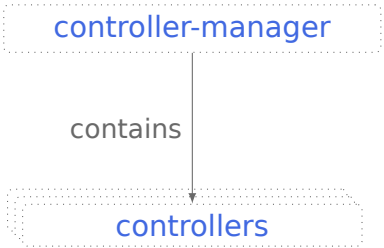
- Watch for pods with an empty PodSpec.NodeName
- Apply (settable) *predicates* to filter out inappropriate nodes
- Apply (built-in) *priority functions* to rank the nodes
  - E.g. the service-pod priority: if a set of pods have a service defined, they should repel each other.
- The node with the highest priority is chosen



Explore the map

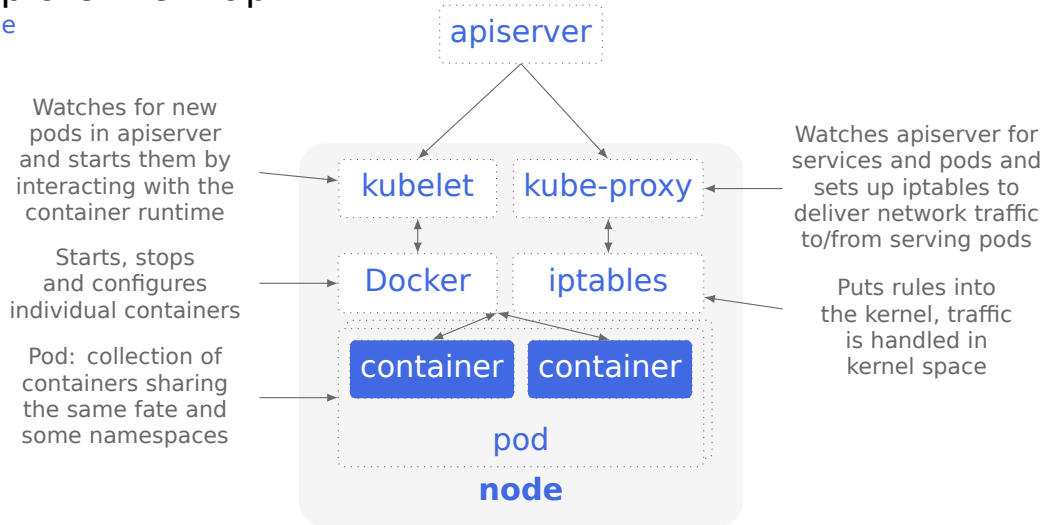
Control plane: controller-manager

- Daemon that embeds the core control loops shipped with Kubernetes
- Example controllers:
  - Replication set
  - Endpoint
  - Namespace
  - Deployment
  - Node
- For example, nodes are discovered, managed and monitored by the node controller



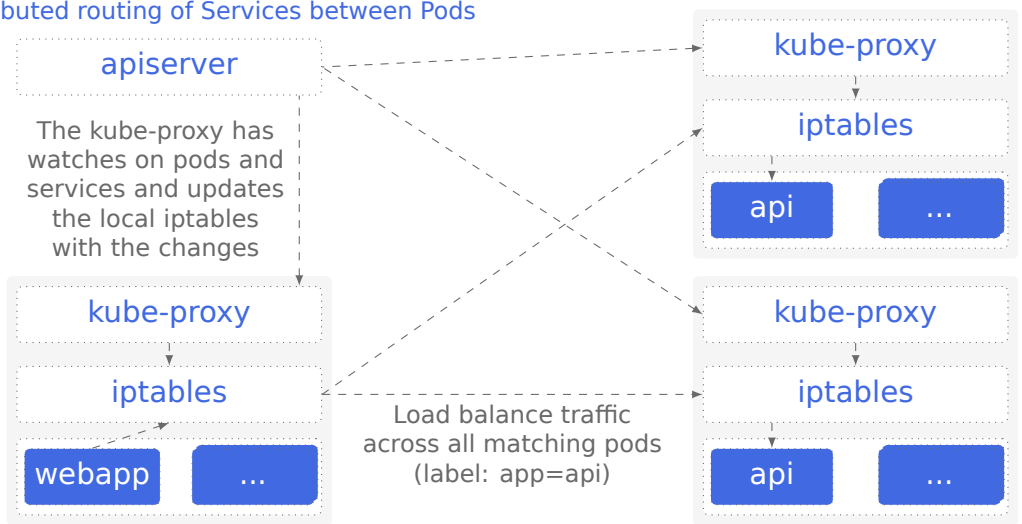
Explore the map

Node



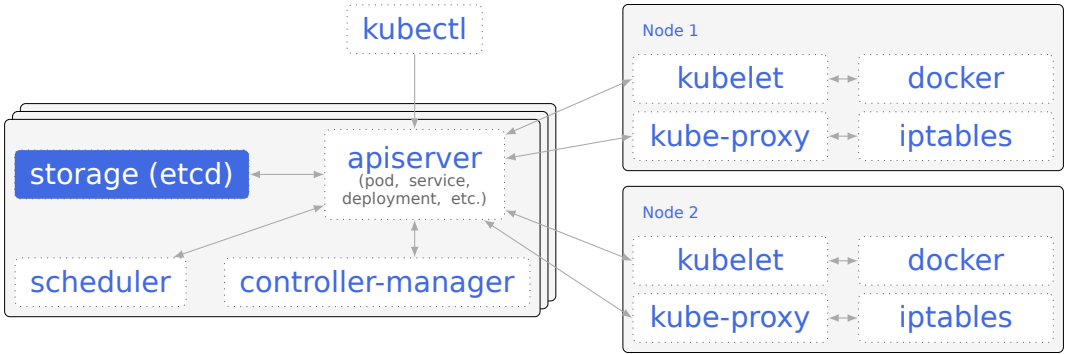
Explore the map

Distributed routing of Services between Pods



Explore the map

Discussion



Discussion: what happens when we do a `$ kubectl create -f newdeployment.yaml`?

Explore the map

Discussion cheatsheet

A quasi correct overview of what happens:

- Local validation in kubectl against pre-cached schemas
- `kubectl` pushes the deployment to the `apiserver` (`~/.kube/config`, TLS)
- API server: authentication (pluggable) + authorization (RBAC) + admission control
  (this happens for every further API server calls too)
- Insert into etcd atomically (or error message)
- `kubectl` returns now (so to get status on further steps, you have to poll)
- Deployment controller: has a watch and therefore notices the new deployment
- Deployment controller: uploads the replicaset to the apiserver
- ReplicaSet controller: has a watch and therefore notices the new replicaset
- Uploads N Pods based on the pod template into the apiserver (with empty nodeName)
- The scheduler has a watch on pods like these
- Scheduler computes the scheduling based on all the predicates and priority functions
- Scheduler patches in the nodeNames of the new pods into the api server
- Kubelets also have watches, so they know about the new pod and asks Docker to start them
- Kubelets then talk to Docker and write back the status to the apiserver (Pending, Running)
- ReplicaSet and Deployment controllers: update the status of their own objects

Useful command for debugging: `kubectl get componentstatuses`


Scheduling
Mechanisms to influence Pod placement

Scheduling

Influence the scheduler's default behavior

Use cases:

- **Isolation**: security, regulatory
- **Different nodes**: GPUs, CPU core difference, network access
- **Latency**: keep the database and the app in the same zone
- **Redundancy**: keep the 3 redundant app servers on different nodes

3 ways to achieve these:

- **Node affinity**: we require/prefer a set of nodes
- **Taints and tolerations**: some nodes have some taints, that some pods can tolerate (e.g. no internet access)
- **Pod (anti-)affinity**: pods can attract and repel each other

Scheduling

Node affinity (used to be nodeSelector)

- Old and simple `nodeSelector` mechanism is deprecated
- Node affinity: pod scheduling rules based on node labels
  - `requiredDuringSchedulingIgnoredDuringExecution`: hard
  - `preferredDuringSchedulingIgnoredDuringExecution`: soft, weights between 1-100; the higher, the more important
  - `requiredDuringExecution`: there are plans for these, but not currently scheduled
- Demo: node labels
- Reference: https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#node-affinity-beta-feature

Scheduling

Node affinity example 1: pod only in zone-{1,2}

```yaml
apiVersion: v1
kind: Pod
metadata: { name: with-node-affinity }
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: failure-domain.beta.kubernetes.io/zone
            operator: In
            values: [ zone-1, zone-2 ]
```

Scheduling

Node affinity example 2: pod prefers a node type

```yaml
apiVersion: v1
kind: Pod
metadata: { name: with-node-affinity }
spec:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: cloud.google.com/gke-nodepool
            operator: In
            values: [ my-node-pool ]
```

Scheduling

Taints and tolerations

- Kubernetes nodes can be *tainted* with particular keys
- This is a kind of anti-affinity, but from the node's direction
- Repel pods from a particular node
- Pods can *tolerate* these taints in order to be scheduled there (e.g. for a pod that's okay with not having internet)
- Taints can have 3 effects:
  - `NoSchedule`: taint node for scheduling
  - `PreferNoSchedule`: soft version of NoSchedule
  - `NoExecute`: taint node for executing, evicting non-tolerate Pods

Scheduling

Automatic taints

Recent versions of K8S are adding more and more automatic taints, e.g.:

- `node.kubernetes.io/unreachable`: no network
- `node.kubernetes.io/memory-pressure`: memory pressure
- `node.kubernetes.io/out-of-disk`: out of disk

Details:
kubernetes.io/docs/concepts/configuration/taint-and-toleration/

Scheduling

Taint example: reserving a Node for ourselves

Tainting a node:

```
$ kubectl taint nodes mynode onlyuser=errge:NoSchedule
```

```yaml
apiVersion: v1
kind: Node
metadata: ...
spec:
  taints:
    - effect: NoSchedule
      key: onlyuser
      value: errge
```

Scheduling

Taint example: tolerating from my debian shell pod

```yaml
apiVersion: v1
kind: Pod
metadata: { name: debian }
spec:
  tolerations:
    - key: onlyuser
      value: errge
      effect: NoSchedule
  containers:
    - name: debian
      image: debian
      command: ["/usr/bin/perl", "-e", "syscall(34)"]
```

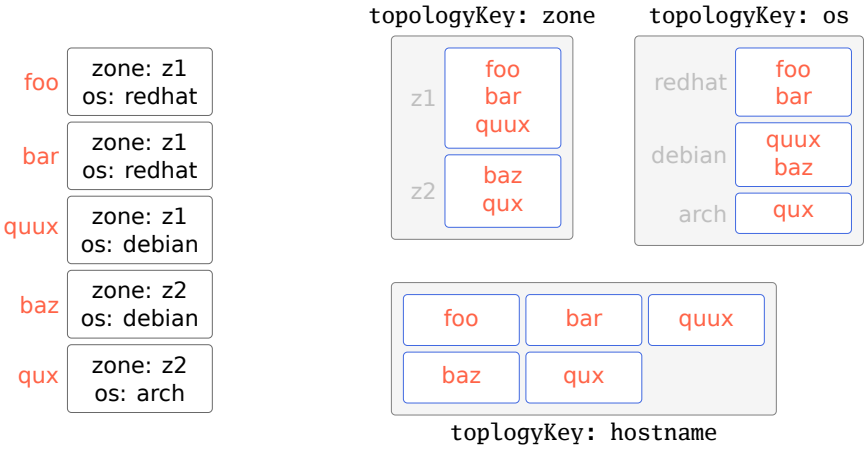
Scheduling

Pod (anti-)affinity

- Scheduling rules based on labels of **other pods**
- `requiredDuringSchedulingIgnoredDuringExecution`: hard
- `preferredDuringSchedulingIgnoredDuringExecution`: soft
- `requiredDuringExecution`: maybe later
- `topologyKey`: first we put nodes into groups by this key
- Pod affinity: have to run in the same group as an other pod
- Pod anti-affinity: can’t run in the same group as an other pod

Scheduling

How do we group nodes by `topologyKey`?



Scheduling

Pod affinity example, latency

- Look at the `topologyKey` and the `labelSelector`
- Will run in a zone where a tier:database labeled pod is running

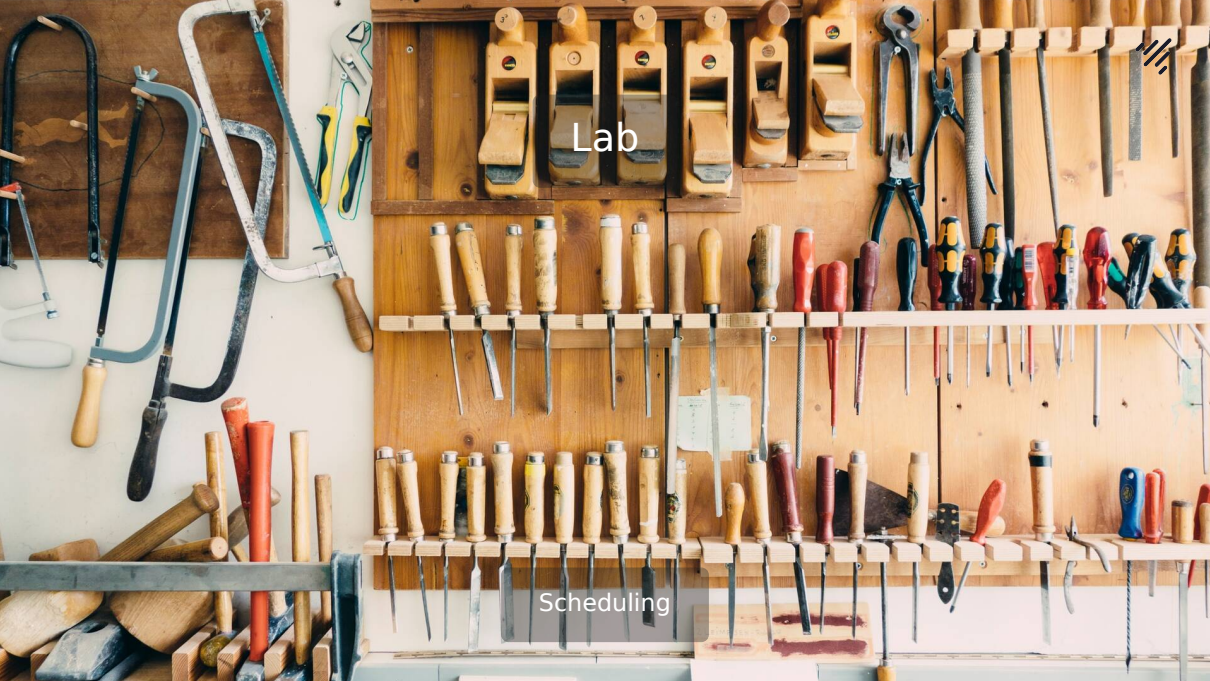
```yaml
kind: Pod
metadata: { name: test, labels: { app: my-app } }
spec:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - topologyKey: failure-domain.beta.kubernetes.io/zone
          labelSelector:
            matchExpressions:
              - key: tier
                operator: In
                values: [ database ]
```

Scheduling

Pod anti-affinity example, redundancy

- Look at the `topologyKey` and the `labelSelector`
- Will prefer to not run on the same node twice

```yaml
kind: Pod
metadata: { name: test, labels: { app: my-app } }
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            topologyKey: kubernetes.io/hostname
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values: [ my-app ]
```


Lab
Scheduling

Lab: scheduling

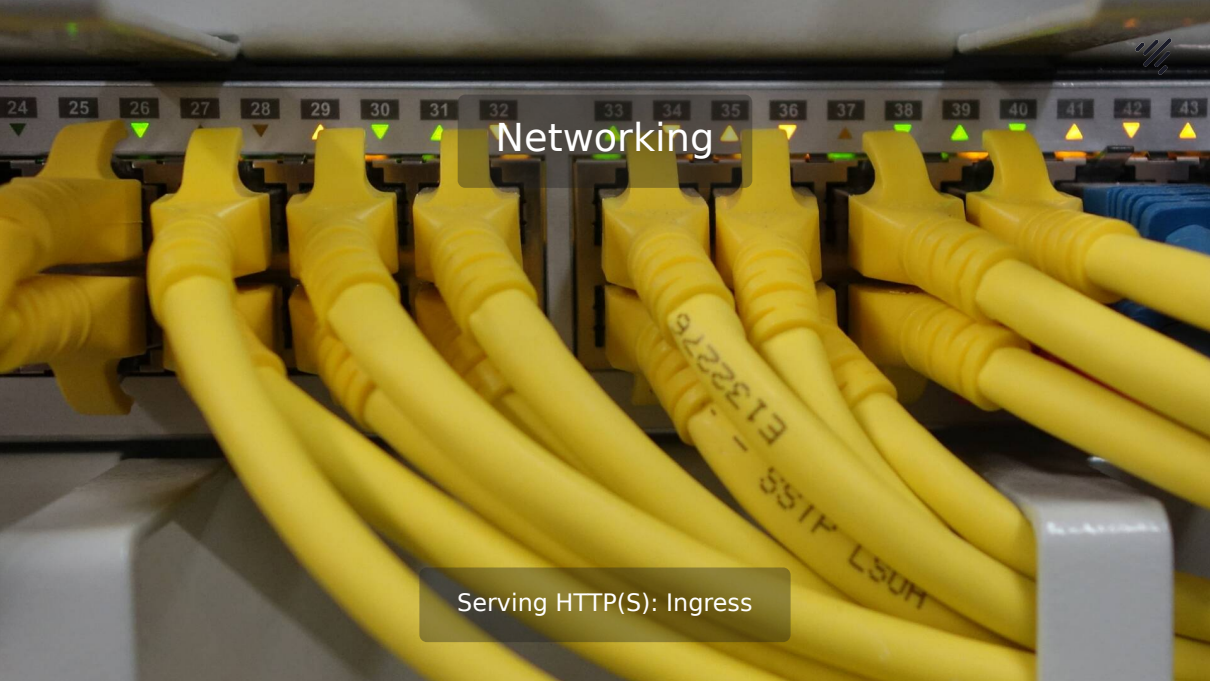
Try out the advanced scheduling features

Try out pod anti-affinity:

- Scale up the front-end deployment
- Add an additional zone to our cluster
- Add a pod anti-affinity rule,
  so the replicas are on different nodes
- Cleanup: go back to one zone

Try out taints and tolerations:

- Ask for a high CPU node, taint it
- Schedule our example pod by tolerating the taint
- Cleanup: delete the pod and the node pool


Networking
Serving HTTP(S): Ingress

Networking

The sock shop story

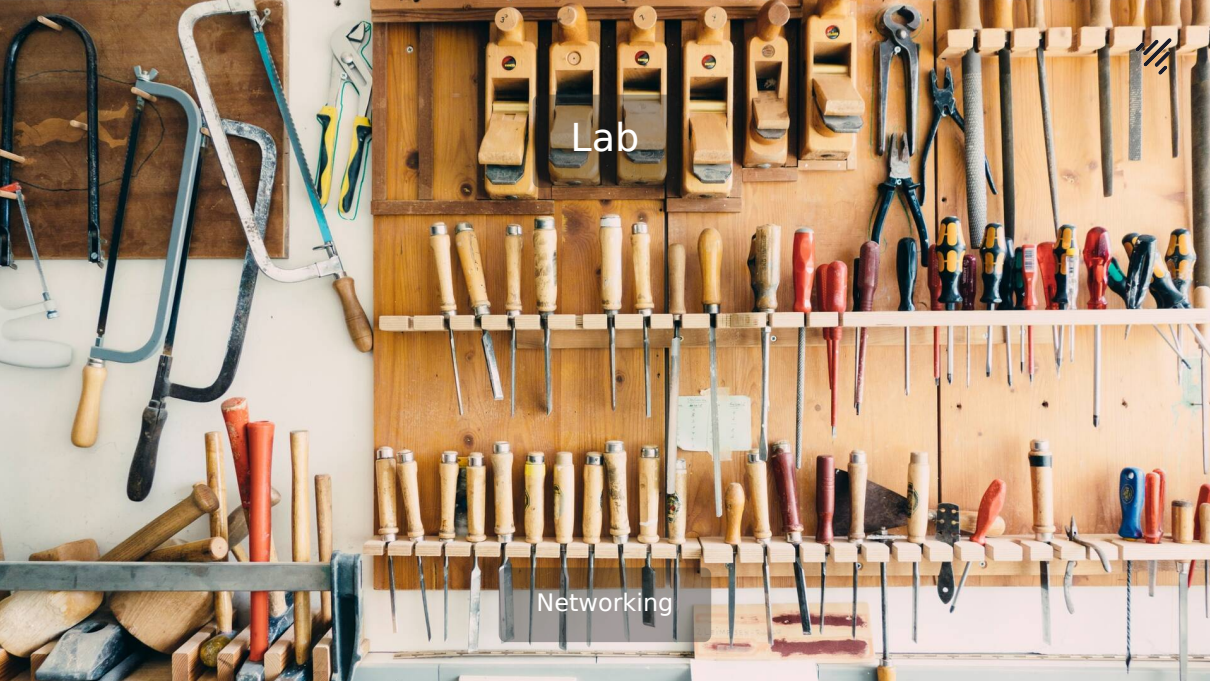
- Our sock shop external IP is currently serving HTTP traffic
- This is of course not acceptable for a production site
- We need HTTPS and we need it now!
- Fortunately we can do this quickly:
  - **Ingress**: higher level abstraction than LoadBalancers, easier to use when the incoming traffic is HTTP(S)
  - **cert-manager**: Jetstack's open-source solution to get HTTPS certificates via Let's Encrypt for free

Ingress

HTTP routing from the internet to your services

- Higher abstraction than services (L7/HTTP vs L4/TCP)
- Not implemented in Kubernetes (but provided by 3rd parties)
- Different implementations (e.g. nginx, GCE)
- Supports hostnames and paths
- SSL certs are easy with cert-manager
- Frequent setup, 2 ingress controllers:
  - GCE: terminate traffic early for nginx
  - nginx: quick reconfiguration

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


Lab
Networking

Lab: networking

Deploy a load-balancing, SSL secured sock shop

- GKE detail: becoming cluster admin
- Deploy nginx-ingress
- Get the external IP and ask the instructor for a domain name
- Create an Ingress (without TLS as a start)
- Remove the old Load Balancer (don't need it anymore)
- Install `cert-manager` to get an TLS certificate
- Use `cert-manager` to get a TLS for our sock shop
- Open the site, you should be automatically redirected to TLS

There are a lot of moving parts, feel free to ask for help!


StatefulSets
Handling Pet services with Kubernetes

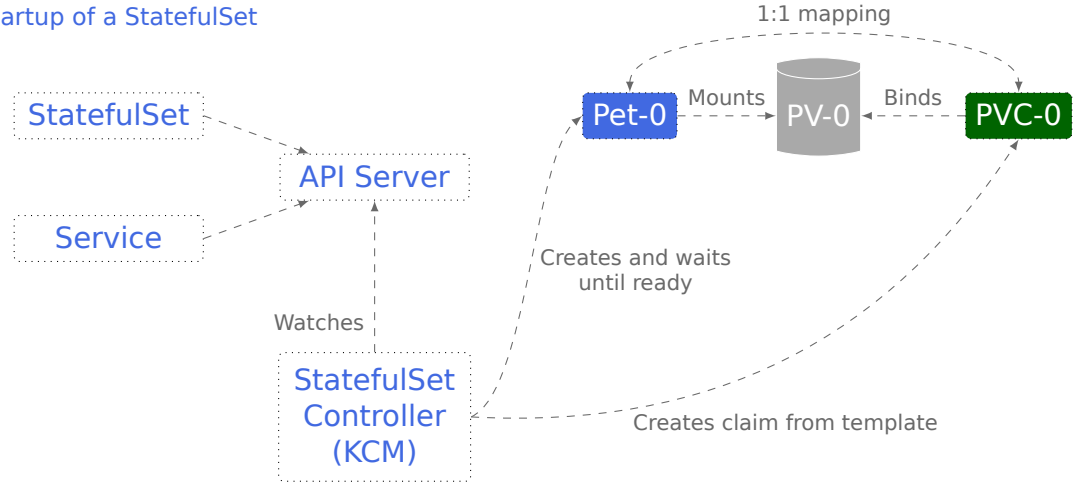
StatefulSets

Different kind of distributed systems

- Some systems are leader-follower based
  (also called master-slave, e.g. etcd)
- In some systems different type of nodes are working together
  (e.g. Elasticsearch)
- StatefulSets handle both: they assign permanent storage for pods
- Our Pods will also have stable hostnames
- Rolling updates will be done more “carefully”
- Storage is based on PVs and PVCs

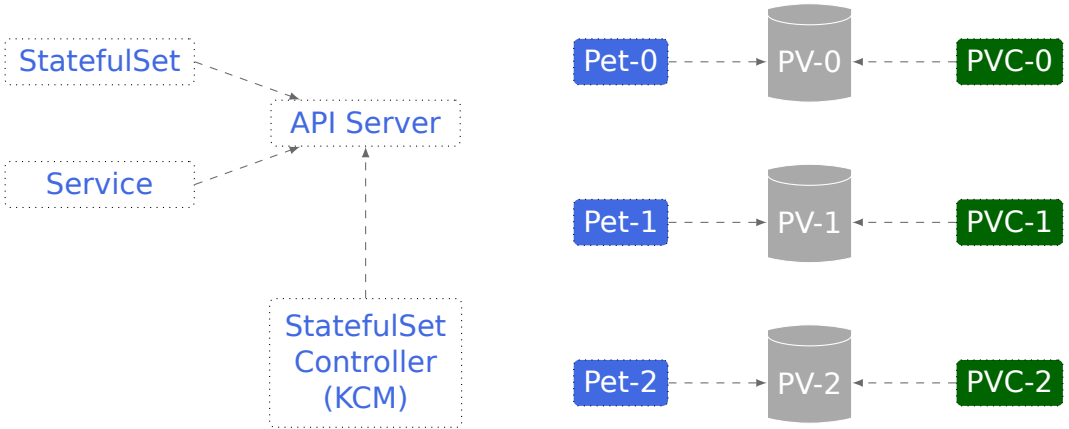
StatefulSets

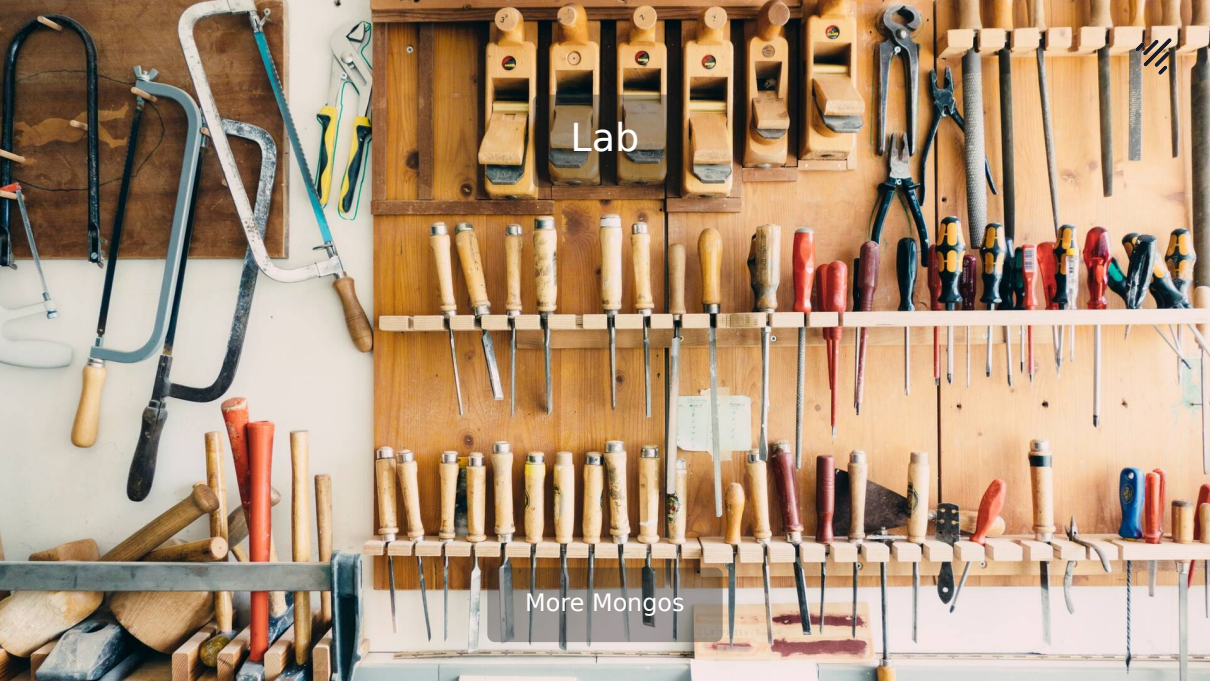
Startup of a StatefulSet



StatefulSets

StatefulSet creation finished




Lab
More Mongos

Lab: more mongos

Deploy a replicated MongoDB for the carts data

- Apply the provided `ssm.yaml` to your cluster
- Check out all the PVs and PVCs that has been created for the pods
- Change the `carts` backend from `carts-db` to the new MongoDBs
- Verify replication: shop in your browser, then kill the master
- Scale up the StatefulSet (MongoDB replica set)
- Bonus: look into how the `ssm.yaml` was generated with Helm


Security
Features for isolation and safety

Security

Security mechanisms available in Kubernetes

- Linux kernel security: namespaces and cgroups
  (used by Docker and Kubernetes)
- API server authentication/authorization: role based (RBAC)
- Admission control: some extra checks after authc/authz
- NetworkPolicy: to limit the any-pod-to-any-pod networking model
- Audit: archive a log of selected actions

Security

API server authentication/authorization

- Authentication is about determining your username and groups
- Many supported authentication methods:
  - X509 client certificates
  - Service account tokens
  - HTTP basic auth (passwords)
  - OAUTH for GKE
  - Other options
- Authorization is about determining what can you do
- Authorization by default is RBAC now
- Roles: definition of what operation on what objects
- Role bindings: definition of who has what roles
- kubernetes.io/docs/reference/access-authn-authz/rbac/

Security

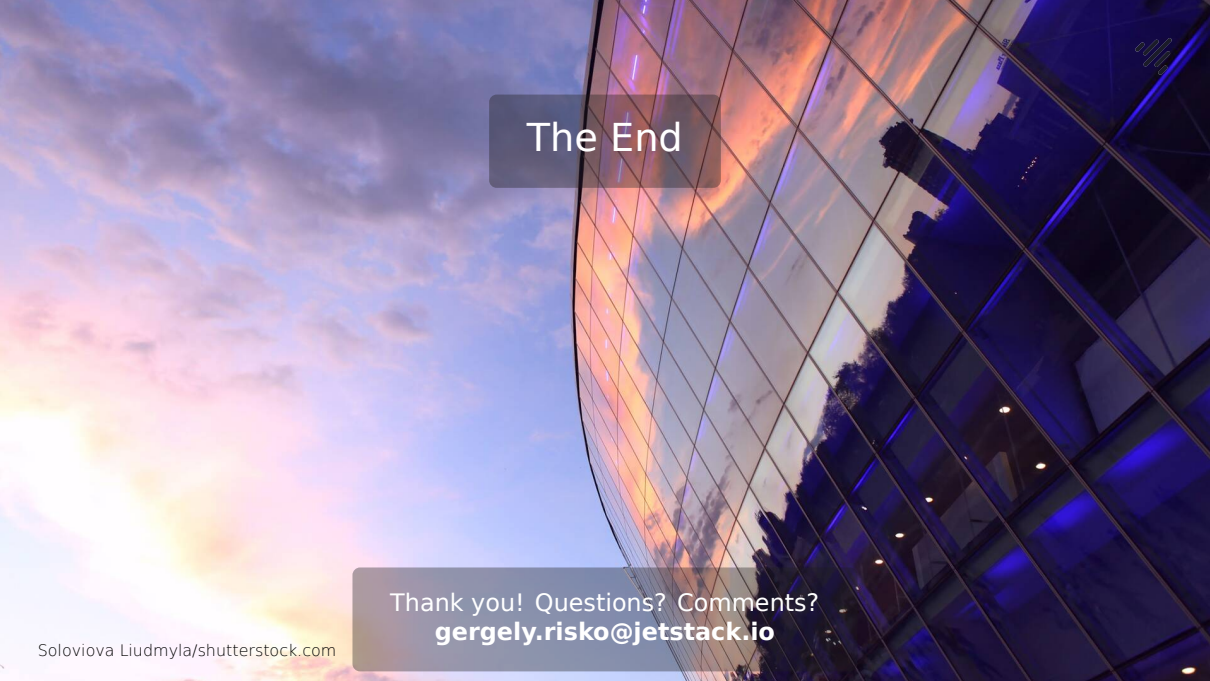
Admission control

- Runs in the API server after authentication and authorization
- But runs before persistence of the object to etcd
- Have to be enabled on the master (so not all usable on GKE)
- Notable admission controllers:
  - `AlwaysPullImages`: forces the image pull policy to Always in every Pod (so people can only access their own Docker images, not hijack something from the cache)
  - `PodSecurityPolicy`: force who can create what kind of Pods on your system (otherwise someone with Pod creation can e.g. run Docker privileged Pods)
- Even with all this knowledge: multi-tenant is a **BAD IDEA**

Further labs

CI/CD and Monitoring

- Monitoring microservices: Prometheus/Grafana, VictoriaMetrics
  - Prometheus: collection and storage of metrics
  - Pull (preferred): microservice has a `/metrics` (e.g. checkout `orders:80/metrics` via a port-forward)
  - Prometheus scrapes these endpoints periodically
  - Push (legacy): pushing metrics to prometheus
  - Grafana: visual graphs for Prometheus queries
  - VictoriaMetrics: long term storage, Prometheus is simple stupid
- CI/CD: making Kubernetes deployments and updates automatic
  - As part of the Git push/review process
  - We use this in Jetstack with GitLab and love it
  - Convenient for developers, and very good for cleanliness



The End

Thank you! Questions? Comments?
gergely.risko@jetstack.io

Soloviova Liudmyla/shutterstock.com