Site Reliability Engineering
-
Containers and Orchestration

- Wolfgang Kandek
wolfgang@kandek.com

Intro

Wolfgang Kandek

- wolfgang@kandek.com
- https://www.linkedin.com/in/wkandek/
- SRE Manager at the Wikimedia Foundation
- Currently migrating from bare-metal to Kubernetes
- Microservices are already on Kubernetes
  - http://wikitech.wikimedia.org/wiki/Kubernetes
- Previously at: Marketo, Google, Qualys, and a couple of startups that do not exist anymore.

Containers and Orchestration

- Containers
- Docker
- Orchestration
- Kubernetes
- Deploying and Scaling Services with Kubernetes
- Kubernetes Security
- Containers Patterns
- Kubernetes Production Patterns
- Service Level Objectives

Existing Knowledge

- Linux
- Incident Response
- Programming

Site Reliability Engineering (SRE)

An SRE team is responsible for the **availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning** of their services

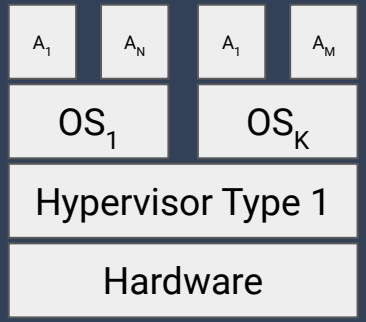
The term was invented at Google, see https://sre.google/sre-book/foreword/

- Engineering - apply engineering technology to computing systems
- Reliability - reliability of those systems is the main objective
- Site/Service - focus on a service

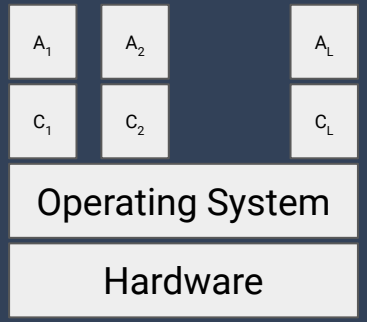
Containers

Our objective as an SRE is to ensure the reliability of an application, of a service. There are a number of infrastructure choices to run the application on. Containers is one possible choice.

Bare Metal



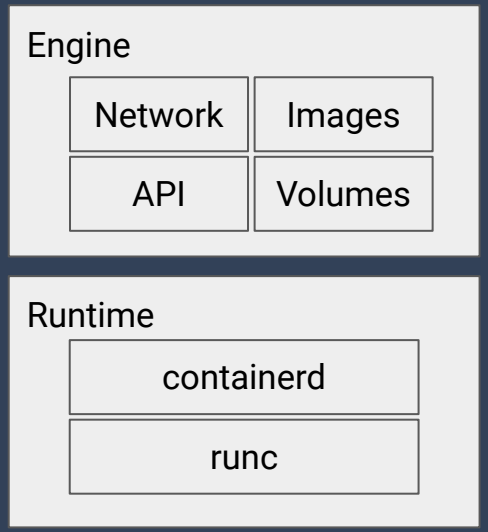
Virtualization



Containers

Docker

Docker is a company that was working on a hosting model using containers. For that they came up with the docker product that we associate with Linux containers. Today they focus completely on that product.
There are two important parts to docker: Engine and Runtime



Docker Runtime

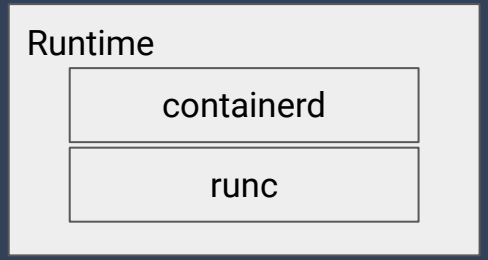
The Docker runtime manages the interaction with the underlying operating system. In the runtime we have 2 components:

- containerd
- runc

Docker uses the Linux features of:

- namespaces - creates new environments for processes, user ids, ...
- cgroups - limits resource usage for CPU, Memory,...

as its foundation for the isolation of multiple running containers.



Docker Runtime

Short Demo for Isolation

Docker Images

Docker manages the building of the container images that end up running your application.

An image contains:

- Application code
- Application dependencies
- Some OS components

Docker Images

Docker manages the building of the container images that end up running your application.

An image contains:

- Application code
- Application dependencies
- Some OS components

And it is organized in layers that are cacheable

```
$ docker pull bitnami/grafana
Using default tag: latest
latest: Pulling from bitnami/grafana
23b664af592e: Already exists
c46241137ba9: Pull complete
ad2e38423d7c: Pull complete
7682bdb036d6: Downloading [====================>                              ]  30.07MB/66.66MB
e19998e5559d: Download complete
4f303e038d6c: Download complete
ae5bbd4fe873: Download complete
cc4f71d17738: Download complete
e8e9007e55ed: Waiting
```

Docker Images

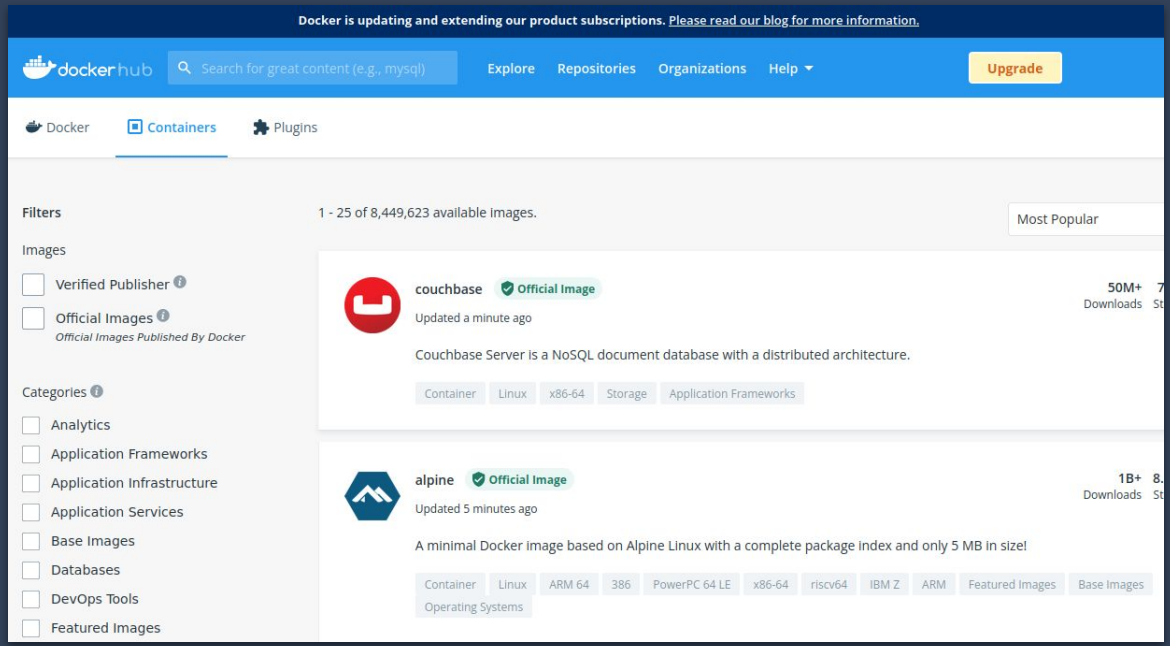
You build your own container image by basing it on an existing one and adding the code and the dependencies of your application.

This example application is written in python and requires the ply and psutil python libraries. The build is run according to a Dockerfile

```
FROM python:latest
COPY server.py /
RUN pip3 install ply psutil
EXPOSE 8080
CMD ["python3", "server.py"]
```

Container Images Repositories

Once developed images need to be stored in a repository to become useful beyond the local machine. Docker manages the most popular repository called Dockerhub at hub.docker.com, host to +/- 8.5 Million images.



Container Images Repositories

There are many categories of images in Dockerhub, for example the official ones: 160+ images curated/built by Docker itself for popular open source projects:

- alpine
- busybox
- ubuntu

- mongodb
- mysql
- postgres
- redis

- httpd
- nginx
- memcached

- node.js
- python
- golang

- hello-world !!

Container Images Repositories

The variety of images available can be a burden.
How can we be sure that we are basing our image on a legitimate build?

Half of 4 Million Public Docker Hub Images Found to Have Critical Vulnerabilities

LIKE DISCUSS

DEC 19, 2020 • 3 MIN READ

by

Hrishikesh Barua FOLLOW

A recent analysis of around 4 million Docker Hub images by cyber security firm Prevasio found that 51% of the images had exploitable vulnerabilities. A large number of these were cryptocurrency miners, both open and hidden, and 6,432 of the images had malware.

Container Images Repositories

The variety of images available can be a burden.
How can we be sure that we are basing our image on a legitimate build?

Half of 4 Million Public Docker H
Vulnerabilities

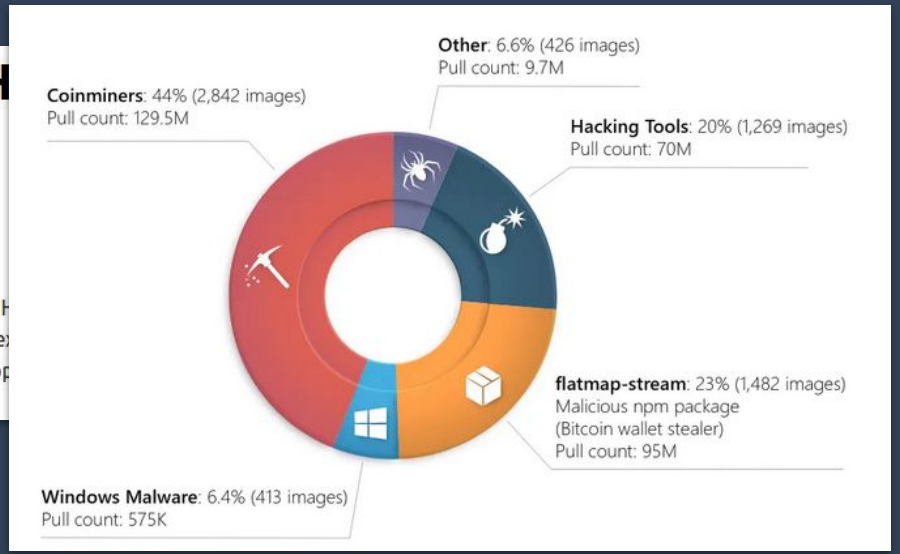
LIKE DISCUSS

DEC 19, 2020 • 3 MIN READ

by

Hrishikesh Barua FOLLOW

A recent analysis of around 4 million Docker H
Prevasio found that 51% of the images had e
of these were cryptocurrency miners, both op
had malware.



Many companies run their own repository for that reason.
Dockerhub offers vulnerability scanning in their paid version. Docker itself has integration for 200 images/month via the docker scan command.

Container Images Repositories

The
How

```
docker scan c1

Testing c1...

✗ Low severity vulnerability found in util-linux/uuid-dev
  Description: Integer Overflow or Wraparound
  Info: https://snyk.io/vuln/SNYK-DEBIAN10-UTILLINUX-1534833
...

✗ Critical severity vulnerability found in glibc/libc-bin
  Description: Integer Overflow or Wraparound
  Info: https://snyk.io/vuln/SNYK-DEBIAN10-GLIBC-1315333

Project name:      docker-image|c1
Docker image:      c1
Platform:          linux/amd64
Base image:        python:3.8

Tested 431 dependencies for known vulnerabilities, found 544 vulnerabilities.

Your base image is out of date
1) Pull the latest version of your base image by running 'docker pull python:3.8'
2) Rebuild your local image

For more free scans that keep your images secure, sign up to Snyk at https://dockr.ly/3ePqVcp
```

,269 images)

,482 images)

Ma
for
Do

Container Lab Walk Through

We will take a look at Lab 1, where we “develop” an application and run it under Docker

Orchestration

Docker itself is of limited use in production. It is a great package format that helps with the discrepancies between development and production environments, but does not address typical production needs:

- Horizontal scaling
- Load Balancing
- Crash Protection
- Tiered Networking
- Resource Control and Optimization
- Security
- Code Rollouts
- ...

Orchestration

To address these production needs we need an Orchestration framework. Docker itself had Docker Swarm, Apache has Mesos and Google reimplemented its internal container system as the open source Kubernetes. Today, Kubernetes (k8s) is the clear leader in the container orchestration field and there are number of companies that support it:

- Open Source k8s - k8s the hard way
- Amazon: Elastic Kubernetes Service (EKS)
- Digital Ocean: Kubernetes
- Google Cloud: Google Kubernetes Engine (GKE)
- Rancher
- VMware Tanzu
- ...

Kubernetes

Kubernetes is an orchestration framework for containerized workloads. It uses declarative configuration and automation.

Kubernetes Concepts and Terms

- Cluster
- Control Plane
- API
- etcd
- Nodes
- Kubelet
- Kube-proxy
- Pods
- Deployment
- Service
- Job
- CronJob
- Kubernetes Client (kubectl)
- ConfigMaps
- Secrets
- RollingUpdates
- Replicas
- Resource Limits
- Probes
- Namespaces

Kubernetes

Deployment: a deployment defines how a container image should be run in Kubernetes. Beyond the container image itself, typical values that are set in a deployment are:

- Number of Replicas
- Network ports in use
- CPU and Memory limits
- Health, Live and Readiness checks
- Attributes/tags
- Rollout mode
- Namespace

Once a deployment is defined in a YAML we can tell K8s to execute it. K8s will then do its best to get the deployment to the defined state and maintain it there, for example always have 3 replicas running, if one crashes restart it, roll out new versions in the specified manner, etc

Kubernetes

Deployment: a deployment un in Kubernetes. Beyond the container image itself, t re:

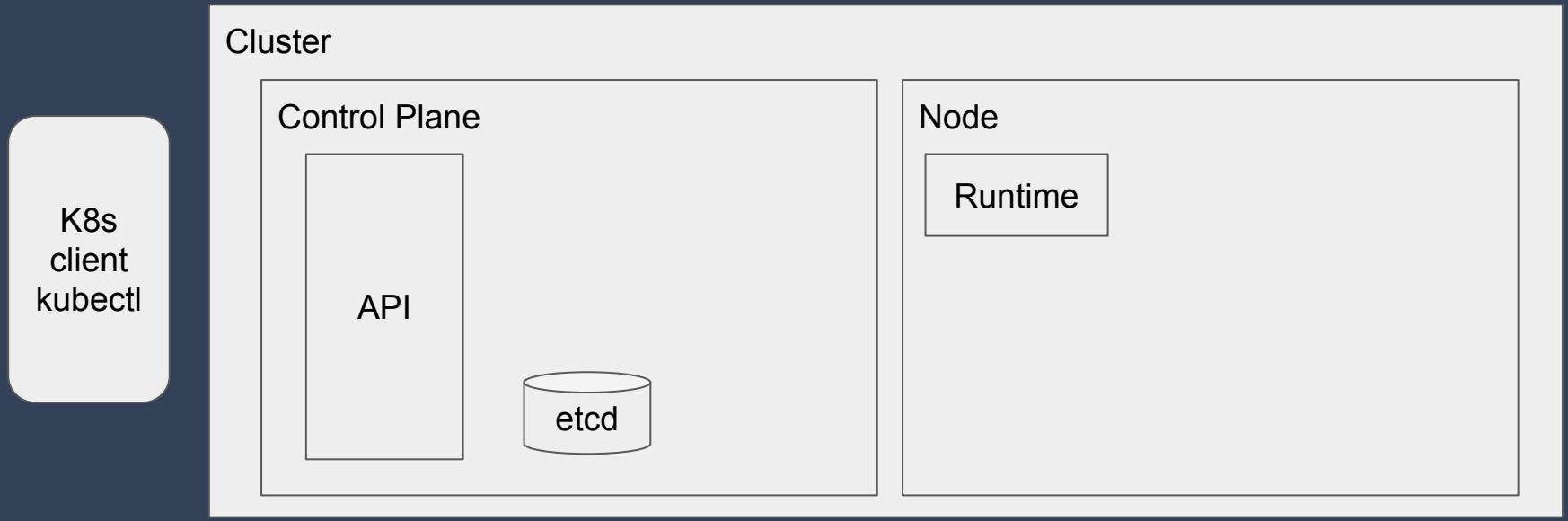
- Number of Replicas
- Network ports in use
- CPU and Memory limi
- Health, Live and Readi
- Attributes/tags
- Rollout mode

Once a deployment is defin K8s will then do its best to get the deployment to th ample always have 3 replicas running, if one cras pecified manner, etc

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: calc
  labels:
    app: calc
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
  selector:
    matchLabels:
      app: calc
  template:
    metadata:
      labels:
        app: calc
    spec:
      containers:
       - name: calc
         image: wkandek/calc:2.0
         imagePullPolicy: Always
         ports:
         - containerPort: 8080
```

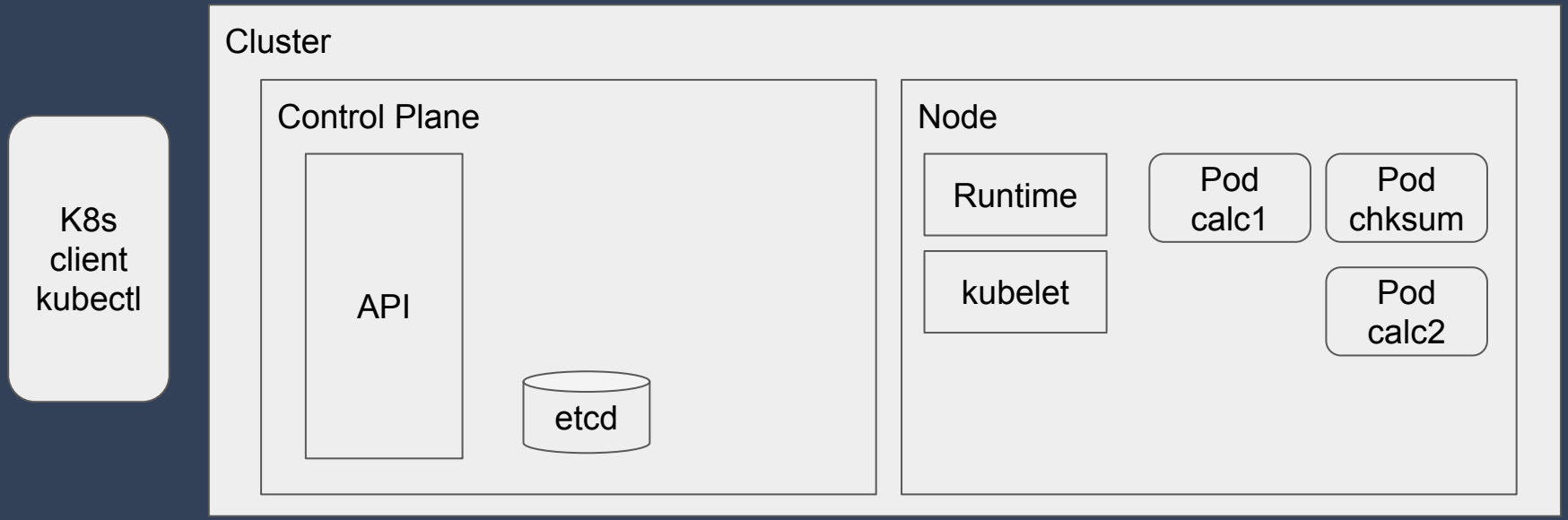
Kubernetes

We tell Kubernetes about our YAML file using the Kubernetes Client, which contacts the API of the cluster. A cluster is composed of the Control Plane, which run the API and the nodes which run the container images. The cluster configuration is stored in the etcd database.



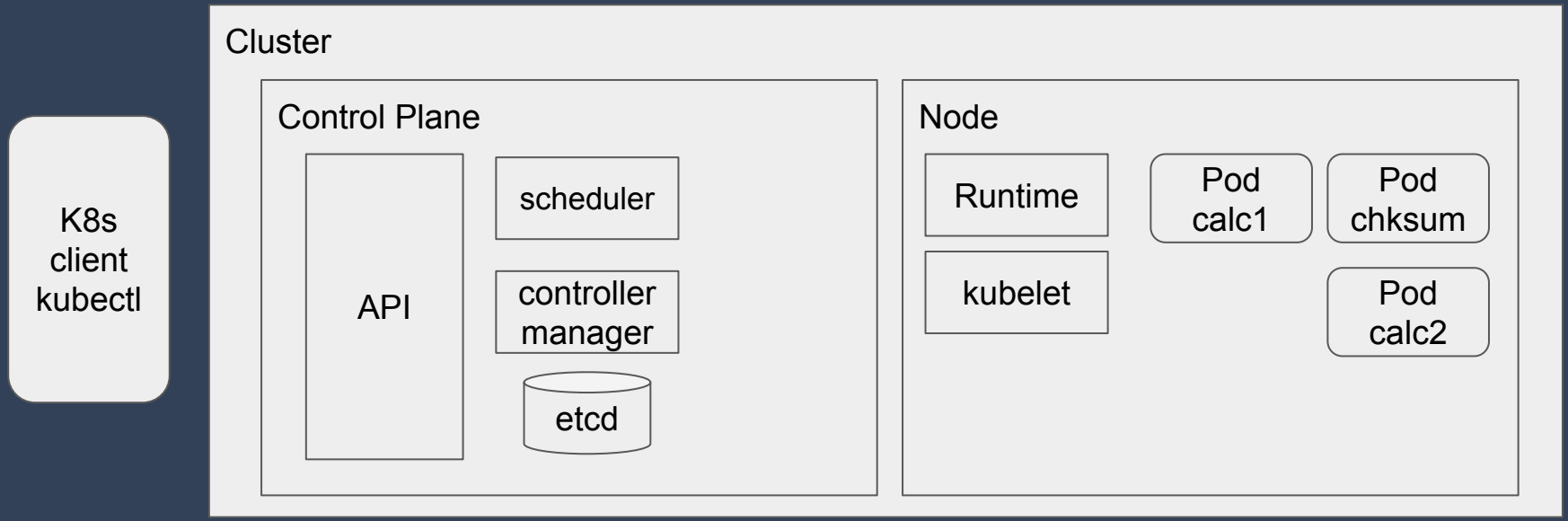
Kubernetes

The container images run in pods. A pod is a group of containers, often just one container. The ControlPlane API server talks bi-directionally using HTTP(S) to the Nodes via the Kubelet that runs on each nodes.



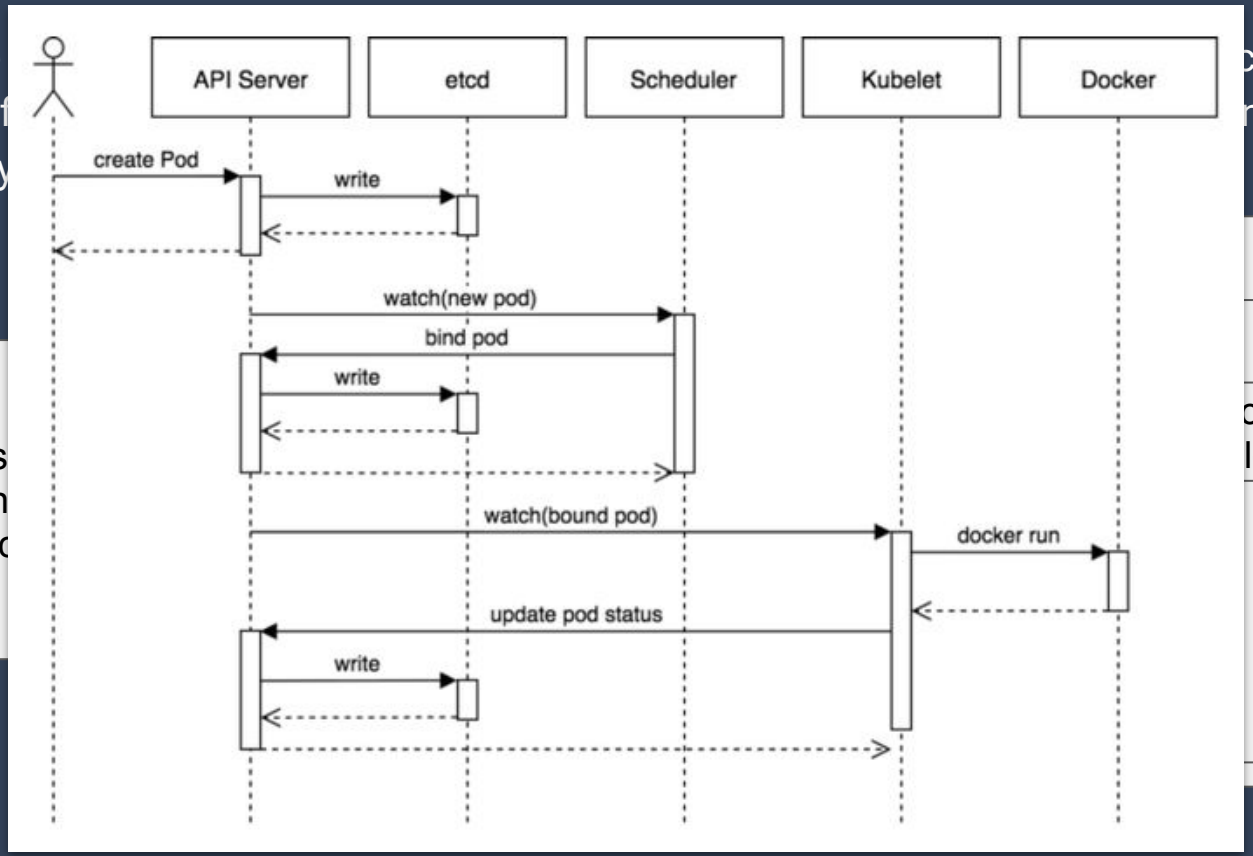
Kubernetes

The Control Plane has a controller and scheduler. The controller manager checks that configuration in etcd and current state are in sync and if not starts the changes that bring them in sync. The scheduler finds the best node for a pod and assigns it.



Kubernetes

The
conf
in sy



checks that
nges that bring them

K8s
clien
kubec

od
lc1

Pod
chksum

Pod
calc2

Kubernetes Networking

A pod gets an internal IP address in the cluster and is reachable by all other pods.

A Service defines how pods can be accessed by internal and external clients. Possible Types:

- ClusterIP - a load balanced, internal IP for a group of pods
  - Inaccessible outside of the cluster
- Nodeport - k8s selects a port, makes it available on all Nodes and forward to a ClusterIP
  - Accessible outside of the cluster on the node level
- Loadbalancer - an external mapping -> Accessible externally

Once a service is defined in a YAML file we can tell K8s to apply it. K8s will then do its best to get the service to the defined state and maintain it there. The service name will also get registered in the k8s internal DNS system.

Kubernetes Networking

A pod gets an internal IP address in the cluster and is reachable by all other pods.

A Service defines how pods can b[illegible] [illegible]ternal clients. Possible Types:

- ClusterIP - a load balanced, i[illegible]
  - Inaccessible outside of
- Nodeport - k8s selects a port[illegible] [illegible]des and forward to a ClusterIP
  - Accessible outside of t[illegible]
- Loadbalancer - an external m[illegible] [illegible]lly

Once a service is defined in a YAM[illegible] [illegible]y it. K8s will then do its best to get the service to the defined state[illegible] [illegible]ervice name will also get registered in the k8s internal DNS

```
kind: Service
apiVersion: v1
metadata:
  name: calc
spec:
  selector:
    app: calc
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Kubernetes Networking

Loadbalancer - an external mapping -> Accessible externally (typically Internet)

Kubernetes does not know how to talk to external load balancers out of the box, but anybody interested can integrate with that setting by writing a controller for it.

The Cloud providers (AWS, Azure, Digital Ocean, GCP for example) have done so and integrated their existing load balancers with their Kubernetes installation and when one defines a Service as Loadbalancer it becomes externally accessible.

MetalLB is one of the products that has provided an integration for Kubernetes as well and it is one of the choices if you are implementing Kubernetes on your own and want to integrate the Loadbalancer option - https://metallb.universe.tf.

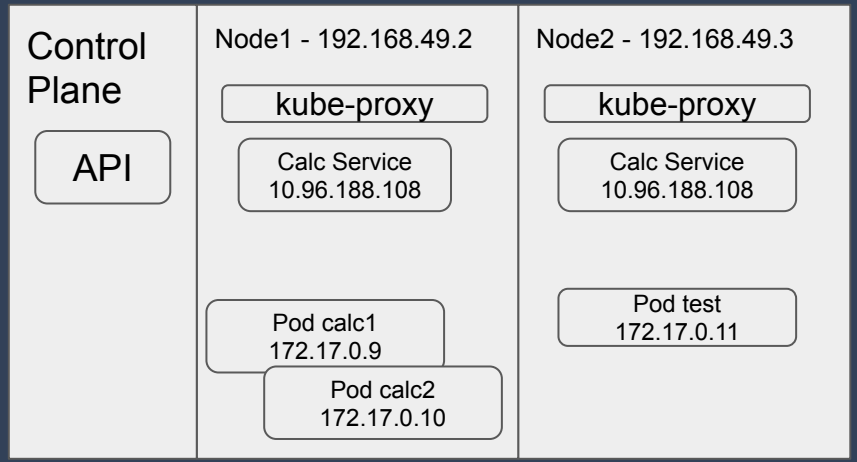
Kubernetes Networking Example 1/4

Each pod gets an IP from the 172.17.x.x network. Each pod can address and reach all other pods on this 172.17.x.x network.

The calc service defined with a type ClusterIP gets an address of 10.96.188.108 which is load balanced across across the pods calc1, calc2.

The kube-proxy process handles load-balancing in the cluster.

From Kubernetes Node 2, pod test can access the calc service individually on pod calc1 and calc2 (all IP 172.17.x.x are routed between nodes by k8s)
or go through the ClusterIP defined by the service calc, which is known on all nodes and load balanced across all pods in the calc deployment

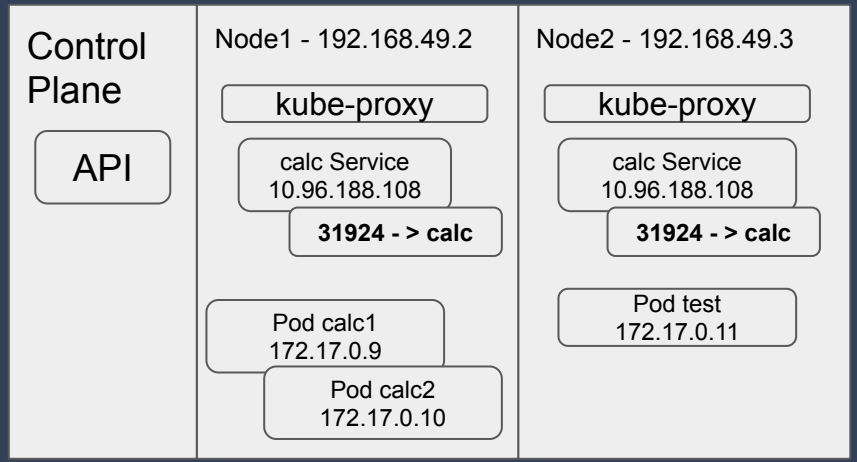


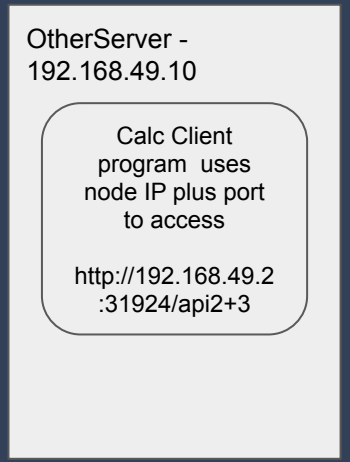
Kubernetes Networking Example 2/4

A server that is in the same network as the nodes, but not in the cluster cannot access the 172.17.x.x or the 10.96.x.x IPs

Instead we need define a NodePort type access where k8s selects a port and makes it available on ALL nodes and forwards it to the service calc

A program on the server OtherServer can now use any Node IP plus the port to access the calc service

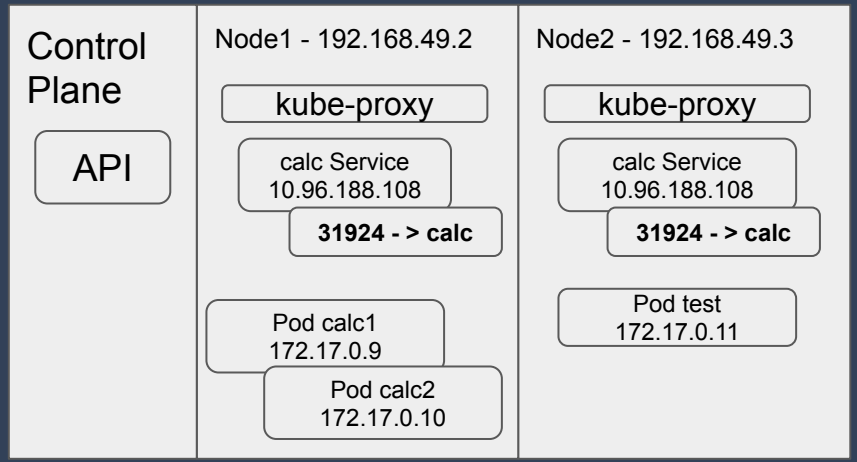


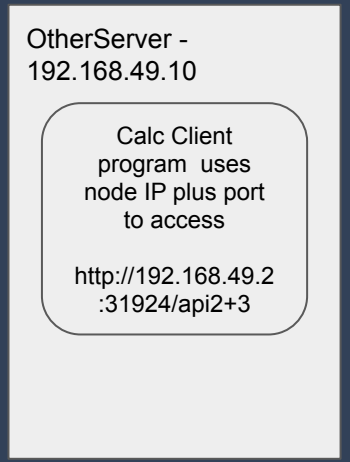


Kubernetes Networking Example 3/4

External non-k8s Loadbalancer
outside Internet (24.17.8.1)
Inside 192.168.49.1
(for calc forward to -
192.168.49.2:31924 and
192.168.49.3:31924)

To give access to the Calc service externally, we can use the NodePort and forward and route relevant traffic to the 2 nodes in the cluster on port 31924. This works with the typical external LoadBalancer (say an F5) in use in many companies

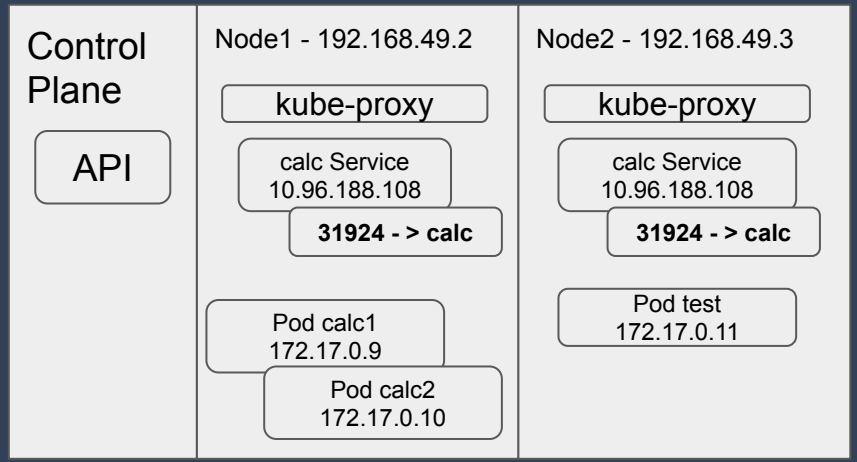


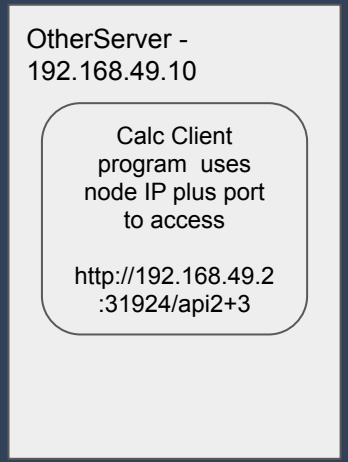


Kubernetes Networking Example 4/4

Integrated Loadbalancer
outside Internet
Inside 192.168.49.x
Automated calc mapping

With an integrated Loadbalancer we can use the Service Type: Loadbalancer and k8s will talk through an installed module to the Loadbalancer and the NodePort mapping will be done automatically





Kubernetes Networking

Kubernetes Networking is essential to the operation of the cluster, but implemented by third party modules (https://kubernetes.io/docs/concepts/cluster-administration/networking/)
Third party networking modules implement a standard called Container Networking Interface (CNI). Some examples:

- Calico
- Cilium
- Flannel
- Weave
- AWS, Azure, GKE

Third party module list: https://kubernetes.io/docs/concepts/cluster-administration/networkingand and some further information: https://platform9.com/blog/the-ultimate-guide-to-using-calico-flannel-weave-and-cilium/

Kubernetes Networking

A pod gets an internal IP address in the cluster and is reachable by all other pods.

A Service defines how pods can b[illegible] [illegible]ternal clients. Possible Types:

- ClusterIP - a load balanced, i[illegible]
  - Inaccessible outside of
- Nodeport - k8s selects a port[illegible]des and forward to a ClusterIP
  - Accessible outside of t[illegible]
- Loadbalancer - an external m[illegible]lly

```
kind: Service
apiVersion: v1
metadata:
  name: calc
spec:
  selector:
    app: calc
  type: LoadBalancer
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
```

Once a service is defined in a YAM[illegible] it. K8s will then do its best to get the service to the defined state[illegible]rvice name will also get registered in the k8s internal DNS

Kubernetes Networking Model is essential to the operation of the cluster, but implemented by third party modules (https://kubernetes.io/docs/concepts/cluster-administration/networking/)

Kubernetes Networking

A pod gets an internal IP address in the cluster and is reachable by all other pods.

```
wkandek:~/projects/k8s/ik/calc$ kubectl apply -f clusterip.yaml
service/calc configured
wkandek:~/projects/k8s/ik/calc$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
calc         ClusterIP   10.108.236.76   <none>        80/TCP    18m
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP   18d
wkandek:~/projects/k8s/ik/calc$ kubectl apply -f nodeport.yaml
service/calc configured
wkandek:~/projects/k8s/ik/calc$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
calc         NodePort    10.108.236.76   <none>        80:30906/TCP     18m
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          18d
wkandek:~/projects/k8s/ik/calc$ minikube ip
192.168.49.2
wkandek:~/projects/k8s/ik/calc$ curl http://192.168.49.2:30906/api?2+3
{"operation":"2+3","result":"5"}wkandek:~/projects/k8s/ik/calc$
```

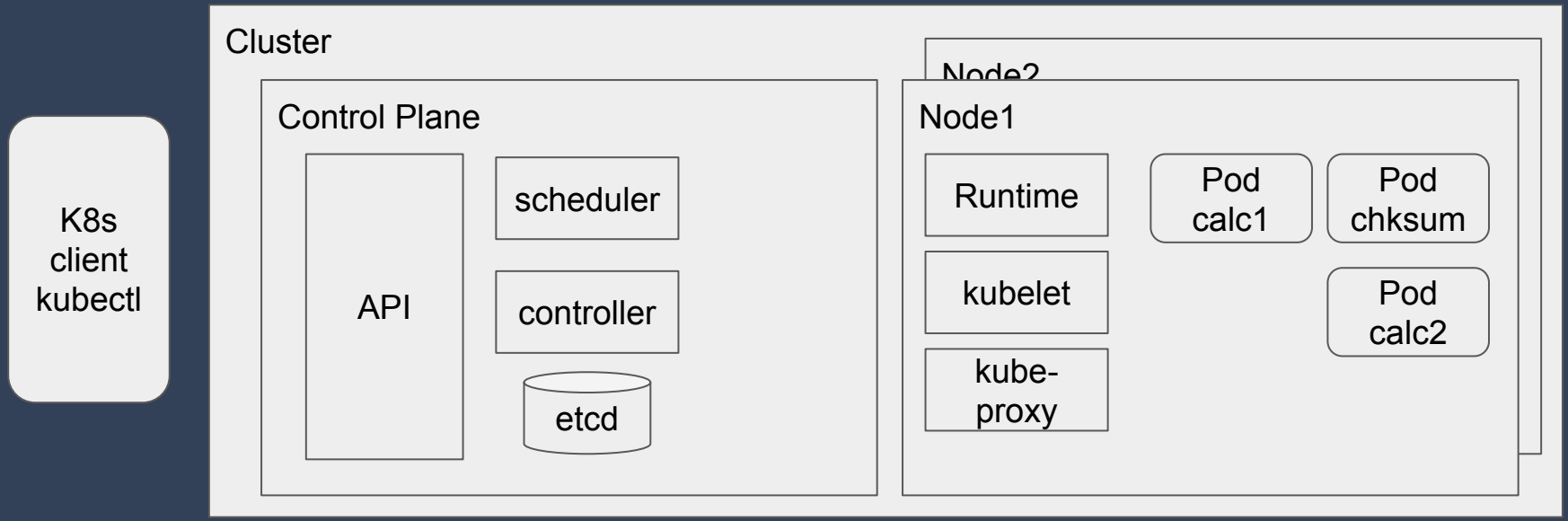
A … Types:

C … best to g…

K … nted by third party modules (https://kubernetes.io/docs/concepts/cluster-administration/networking/)

Kubernetes Networking

The kube-proxy on each node is responsible for the networking part of a service. For example if we have 3 replicas in the calc deployment, 2 on Node1 and 1 on Node2, kube-proxy will make sure that they get load balanced traffic.



Kubernetes Networking - Ingress

An Ingress controller is networking component that sits above the service layer. It acts as a HTTP/HTTPS router and can forward based traffic based on a path. For example /calculator/ can be routed to the calc service, whereas /hpa/ can be routed to the service for the demo application of the horizontal pod autoscaler.

Ingress controllers do not come with Kubernetes, but instead are implemented by 3rd parties. Examples:

- haproxy
- nginx
- traefik
- ...

Capabilities vary by controller, but typically include SSL termination and load balancing.

Kubernetes Networking - Ingress

An Ingress cont rvice layer. It acts as a
HTTP/HTTPS ro For example /calculator/
can be routed to service for the demo
application of th

Ingress can also

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: calc-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: ingress.info
      http:
        paths:
          - path: /calculator/(.*)
            pathType: Prefix
            backend:
              service:
                name: calc
                port:
                  number: 80
          - path: /hpa/(.*)
            pathType: Prefix
            backend:
              service:
                name: hpa
                port:
                  number: 80
```

Kubernetes Storage

A container in a pod has its own disk storage. It is ephemeral. When modified the modifications are lost on restart.

Kubernetes allows for the definition of persistent storage. Persistent storage is used for data that needs to survive a pod restart. Kubernetes does not implement persistent storage natively but relies on third party modules. These modules implement a standard called Container Storage Interface (CSI).

Examples:

- AWS EBS, Azure, GCE
- Ceph, NFS, gluster
- Local, hostpath, emptyDir

Kubernetes Lab Walk Through

We will take a look at Lab 2, where we deploy an application to a local Kubernetes

Kubernetes Namespaces

Kubernetes provides for a separate namespace for a deployment. A namespace isolates and groups applications and helps in their management. Imagine a k8s cluster with 1000s of pods of different applications running in the default namespace, it is much more usable to just list wand work with the pods for a certain namespace (kubectl get pods -n calc).

It is considered best practice to create namespaces for an application or a group of applications.

Kubernetes

ConfigMaps: a way to pass configuration values to the application in a pod.

- The server to access to a certain service, a hostname for example
- Stored in Cleartext

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: bookapp-configmap
data:
  database: bookdb
```

Kubernetes

ConfigMap: a way to pass configuration values to the application in a pod.

- The server to access to a certain service, a hostname for example
- Stored in Cleartext

Secret: a way to pass confidential information to a pod

- A password or API key to access a service
- Stored in base64 encoding

```
apiVersion: v1
kind: Secret
metadata:
  name: bookdb-credentials
type: Opaque
data:
   db-password: Ym9va2RicGFzcw==
   db-username: Ym9va2Ri
```

Kubernetes

ConfigMaps: a way to pass configuration values to the application in a pod.

- The server to access to a certain service, a hostname for example
- Stored in Cleartext

Secrets: a way to pass confidential information to a pod

- A password or API key to access a service
- Stored in base64 encoding

Both do not require a new image, but just a new deployment. This means they are light-weight in terms of system resources and fast.

Kubernetes Health Checks and Probes

Readiness probe: a way to check whether a service is ready to serve requests, used for load balancing

- Has an initial wait time, a cycle time and success and failure thresholds

Startup probe: a way to account for a service that requires a longer than normal startup time after it was started

- Imagine having to load data and compute a set of rules
- Has priority over Readiness

Liveness probe: a way to check whether the pod is healthy

- Unhealthy pods will be restarted

Probes are defined in the deployment YAML and can use HTTP requests, TCP connections or a command. The kubelet on the node runs the checks.

Kubernetes Health Checks and Probes

Readiness probe: a way to ...ests, used for load balancing

- Has an initial wait tim...olds

Startup probe: a way to acc...normal startup time after it was started

- Imagine having to loa...
- Has priority over Reac...

Liveness probe: a way to ch...

- Unhealthy pods will b...

```yaml
spec:
  containers:
  - name: calc
    image: wkandek/calc:2.0
    imagePullPolicy: Always
    ports:
    - containerPort: 8080
    readinessProbe:
      httpGet:
        scheme: HTTP
        path: /metrics
        port: 8080
      initialDelaySeconds: 10
      periodSeconds: 5
```

Probes are defined in the deployment YAML and can use HTTP requests, TCP connections or a command.The kubelet on the node runs the checks.

Kubernetes Health Checks and Probes

```
wkandek:~/projects/k8s/ik/liveness$ cat Dockerfile
FROM ubuntu
ADD wait_then_exit.sh /
ENTRYPOINT ["/bin/bash","/wait_then_exit.sh"]
wkandek:~/projects/k8s/ik/liveness$ cat wait_then_exit.sh
#/bin/bash
touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
```

sts, used for load

s

rmal startup time

- Imagine having to load data and compute a set of rules
- Has priority over Readiness

Liveness probe: a way to check whether the pod is healthy

- Unhealthy pods will be restarted

Probes are defined in the deployment YAML and can use HTTP requests, TCP connections or a command. The kubelet on the node runs the checks.

Kubernetes Health Checks and Probes

```
wkandek:~/projects/k8s/ik/liveness$ cat Dockerfile
FROM ubuntu
ADD wait_then_exit.sh /
ENTRYPOINT ["/bin/bash","/wait_then_exit.sh"]
```

sts, used for load

s

```
wkandek:~/projects/k8s/ik/liveness$ kubectl run waittest --image=wkandek/wait:1.0 --restart='Never'
pod/waittest created
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods waittest
NAME       READY   STATUS    RESTARTS   AGE
waittest   1/1     Running   0          12s
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods waittest
NAME       READY   STATUS    RESTARTS   AGE
waittest   1/1     Running   0          6m38s
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods waittest
NAME       READY   STATUS    RESTARTS   AGE
waittest   1/1     Running   0          10m
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods waittest
NAME       READY   STATUS      RESTARTS   AGE
waittest   0/1     Completed   0          10m
```

Probes are defined in the deployment YAML and can use HTTP requests, TCP connections or a command. The kubelet on the node runs the checks.

Kubernetes Health Checks and Probes

```
wkandek:~/proje
FROM ubuntu
ADD wait_then_e
ENTRYPOINT ["/b
wkandek:~/proje
#/bin/bash
touch /tmp/heal
```

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: wkandek/wait:1.0
    imagePullPolicy: Always
    resources: {}
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

```
le

n_exit.sh

y; sleep 600
```

sts, used for load

s

rmal startup time

- Imagine ha
- Has priority

es

Liveness probe:

- Unhealthy p

Probes are defin

TTP requests, TCP connections or a command. The k

Kubernetes Health Checks and Probes

Readiness
balancing
- Has

Startup pr
after it wa
- Imag
- Has

Liveness p
- Unhe

```
wkandek:~/projects/k8s/ik/liveness$ kubectl apply -f d.yaml
pod/liveness-exec created
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
liveness-exec   1/1     Running   0          6s
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods
NAME            READY   STATUS    RESTARTS      AGE
liveness-exec   1/1     Running   1 (17s ago)   92s
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods
NAME            READY   STATUS    RESTARTS      AGE
liveness-exec   1/1     Running   2 (58s ago)   3m28s
wkandek:~/projects/k8s/ik/liveness$ kubectl get pods
NAME            READY   STATUS    RESTARTS      AGE
liveness-exec   1/1     Running   3 (37s ago)   4m22s
```

Probes are defined in the deployment YAML and can use HTTP requests, TCP connections or a command. The kubelet on the node runs the checks.

Kubernetes Health Checks and Probes

Readiness probe: a way to check whether a service is ready to serve requests, used for load balancing

- Has an initial wait time, a cycle time and success and failure thresholds

Startup probe: a way to account for a service that requires a longer than normal startup time after it was started

```
Events:
  Type     Reason     Age                  From               Message
  ----     ------     ----                 ----               -------
  Normal   Scheduled  <unknown>                               Successfully assigned default/liveness-exec to minikube
  Normal   Pulled     2m34s                kubelet, minikube  Successfully pulled image "wkandek/wait:1.0" in 11.482892983s
  Normal   Pulled     79s                  kubelet, minikube  Successfully pulled image "wkandek/wait:1.0" in 1.258067423s
  Warning  Unhealthy  36s (x6 over 2m1s)   kubelet, minikube  Liveness probe failed: cat: /tmp/healthy: No such file or directory
  Normal   Killing    36s (x2 over 111s)   kubelet, minikube  Container liveness failed liveness probe, will be restarted
  Normal   Pulling    6s (x3 over 2m45s)   kubelet, minikube  Pulling image "wkandek/wait:1.0"
```

Probes are defined in the deployment YAML and can use HTTP requests, TCP connections or a command. The kubelet on the node runs the checks.

Kubernetes Health Checks and Probes

- Readiness probe: a way to check whether a service is ready to serve requests, used for load

```
wkandek:~/projects/k8s/liveness$ kubectl describe pod liveness-exec
Name:         liveness-exec
Namespace:    default
Priority:     0
Node:         minikube/192.168.49.2
Start Time:   Sat, 23 Oct 2021 18:30:03 -0700
Labels:       test=liveness
Annotations:  Status:  Running
IP:           172.17.0.3

Events:
  Type     Reason     Age                   From               Message
  ----     ------     ----                  ----               -------
  Normal   Scheduled  <unknown>                                Successfully assigned default/liveness-exec to minikube
  Normal   Pulled     18m                   kubelet, minikube  Successfully pulled image "k8s.gcr.io/busybox" in 1.410834422s
  Normal   Pulled     17m                   kubelet, minikube  Successfully pulled image "k8s.gcr.io/busybox" in 920.143234ms
  Normal   Created    15m (x3 over 18m)     kubelet, minikube  Created container liveness
  Normal   Started    15m (x3 over 18m)     kubelet, minikube  Started container liveness
  Normal   Pulled     15m                   kubelet, minikube  Successfully pulled image "k8s.gcr.io/busybox" in 809.893358ms
  Warning  Unhealthy  15m (x9 over 17m)     kubelet, minikube  Liveness probe failed: cat: can't open '/tmp/healthy': No such file or directory
  Normal   Killing    15m (x3 over 17m)     kubelet, minikube  Container liveness failed liveness probe, will be restarted
  Normal   Pulling    14m (x4 over 18m)     kubelet, minikube  Pulling image "k8s.gcr.io/busybox"
  Normal   Pulled     13m                   kubelet, minikube  Successfully pulled image "k8s.gcr.io/busybox" in 910.464656ms
  Warning  BackOff    3m16s (x28 over 10m)  kubelet, minikube  Back-off restarting failed container
```

Probes are defined in the deployment YAML and can use HTTP requests, TCP connections or a command. The kubelet on the node runs the checks.

Kubernetes Resources and Limits

Containers should have resource limits specified. Kubernetes will limit the containers to the specified limits. During startup Kubernetes will use the limits to select the best nodes for the containers

- CPU: in fractions of a Kubernetes CPUs = 1 core or 1 hyperthread
    - 0.1, 0.5, 1.5, 2 ...
- Memory: in Bytes, 64 Ki, 256 Mi, 1 Gi ... - also K,M,G…
    - Mi = Megabytes (2^20) whereas M = 10^6, which is a bit smaller
- Local Storage (Ephemeral) : 4 Gi
- Others...

A pod that uses too much CPU might get throttled. A pod that uses too much memory might be killed. It will then be restarted if marked as restartable.

Request vs Limit: Request is the minimum needed, Limit the maximum.

Kubernetes Resources and Limits

Containers should have resource limits specified. Kubernetes will limit the containers to the specified limits. During startup Kubernetes will use the limits to select the best nodes for the containers

```
spec:
  containers:
  - name: calc
    image: wkandek/calc:2.0
    imagePullPolicy: Always
    ports:
    - containerPort: 8080
    resources:
      limits:
        cpu: "1"
      requests:
        cpu: "0.5"
```

- CPU: in fractions … hread
  - 0.1, 0.…
- Memory: in Bytes…
  - Mi = M… s a bit smaller
- Local Storage (E…
- Others...

A pod that uses too m… too much memory might be killed. It will then be restarted if marked as restartable.

Request vs Limit: Request is the minimum needed, Limit the maximum.

Kubernetes Horizontal Pod Autoscaler

Kubernetes Horizontal Pod Autoscaler (HPA) can be used to automatically increase (and decrease) the number of replicas of a pod to deal with fluctuating load.

- Minimum and Maximum number of replicas
- Resource Types
  - CPU
  - Pod metrics
  - Object Metrics

Scale up and down are not immediate but require some time of usage above limits

Kubernetes Horizontal Pod Autoscaler

Kubernetes Horizontal Pod Autoscaler (HPA) can be used to automatically increase (and decrease) the number of replicas of a pod to deal with fluctuating load.

- Min
- Res
  -
  -
  -

Scale up ... bove limits

```
wkandek:~/projects/k8s/k8s-demo/hpa$ kubectl get hpa
NAME   REFERENCE         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa1   Deployment/hpa1   0%/50%    1         10        1          54m
wkandek:~/projects/k8s/k8s-demo/hpa$ kubectl get hpa
NAME   REFERENCE         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa1   Deployment/hpa1   0%/50%    1         10        1          55m
wkandek:~/projects/k8s/k8s-demo/hpa$ kubectl get hpa
NAME   REFERENCE         TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
hpa1   Deployment/hpa1   328%/50%   1         10        4          55m
wkandek:~/projects/k8s/k8s-demo/hpa$ kubectl get hpa
NAME   REFERENCE         TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
hpa1   Deployment/hpa1   328%/50%   1         10        7          55m
wkandek:~/projects/k8s/k8s-demo/hpa$ kubectl get hpa

NAME   REFERENCE         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa1   Deployment/hpa1   42%/50%   1         10        7          60m
wkandek:~/projects/k8s/k8s-demo/hpa$
wkandek:~/projects/k8s/k8s-demo/hpa$ kubectl get hpa
NAME   REFERENCE         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa1   Deployment/hpa1   45%/50%   1         10        8          66m
wkandek:~/projects/k8s/k8s-demo/hpa$ kubectl get hpa
NAME   REFERENCE         TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
hpa1   Deployment/hpa1   0%/50%    1         10        1          81m
```

Kubernetes Vertical Pod Autoscaler

Kubernetes Vertical Pod Autoscaler (VPA) is an installable option for Kubernetes. It is used to automatically change the size of the pods in the cluster to their observed usage. It adjusts the CPU and Memory request settings in a deployment.

See: https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler
and
https://medium.com/infrastructure-adventures/vertical-pod-autoscaler-deep-dive-limitations-and-real-world-examples-9195f8422724

Kubernetes Cluster Autoscaler

Kubernetes Cluster Autoscaler is an installable option for Kubernetes. It is used to automatically increase and decrease the size of the cluster by adding and deleting nodes. It is dependent on your hardware environment:

- AWS
- Azure
- Digital Ocean
- GCP - also has an AutoPilot GKE Option
- OpenShift

Scale up and down are not immediate but require some time of usage above limits.
https://github.com/kubernetes/autoscaler/tree/master/cluster-autoscaler

Kubernetes Jobs and CronJobs

Kubernetes Jobs are for batch runs. They just run a container image once. There are some new features:

- parallelism
- indexing

CronJobs are similar to Unix/Linux cronjobs: you can program them to run periodically.

Kubernetes improved on the Unix/Linux design:

- concurrencyPolicy: allow/forbid/replace
- activeDeadlineseconds: maximum runtime

Issues:

- Scheduling latency/Image pull latency (startingDeadlineseconds field exists)
- CPU/Memory Resources not available
- Node problems

Kubernetes RollingUpdates

Kubernetes supports the installation of a new version of a container image by providing a number of rolling update mode. Kubernetes will start creating new pods, then stop routing traffic to a set of old pods. By default Kubernetes will keep 75% of nodes functioning at all points in time.

- Kubectl rollout command
    - status
    - history
    - undo
    - pause/resume

Kubernetes Packages

Kubernetes is driven by YAML files that should be stored in a version control system. Nevertheless it can be difficult to control all the files involved in a code rollout:

- deployment
- service
- configmaps
- secrets
- ...

In software development we bundle files that belong together in a package. Under Kubernetes we can do similar things with a package manager. Helm is the most popular tool for that purpose and it provides functionality for many common use cases.

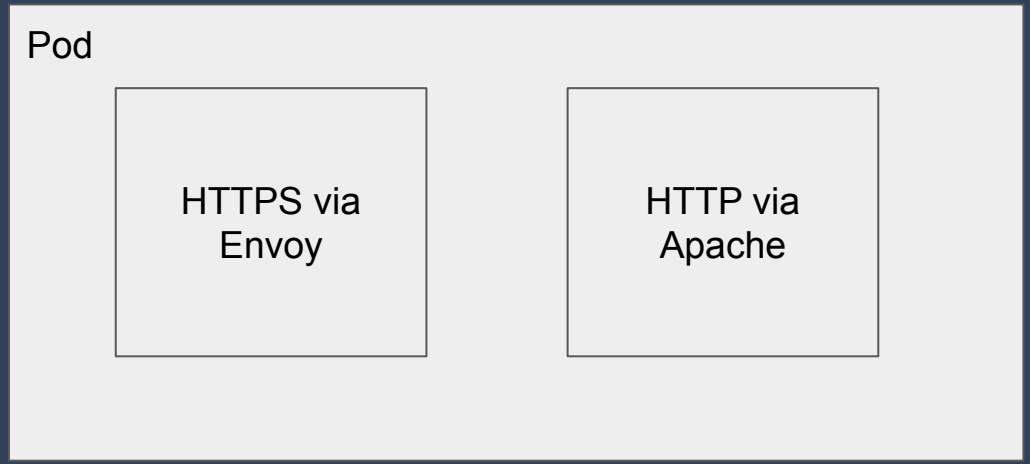
Kubernetes Lab Walk Through

We will take a look at Lab 3, where we deploy an application to a managed Kubernetes

Sidecar Pattern

We can put more than one container in a pod, all of them will be scheduled and run concurrently and they share the network address space.

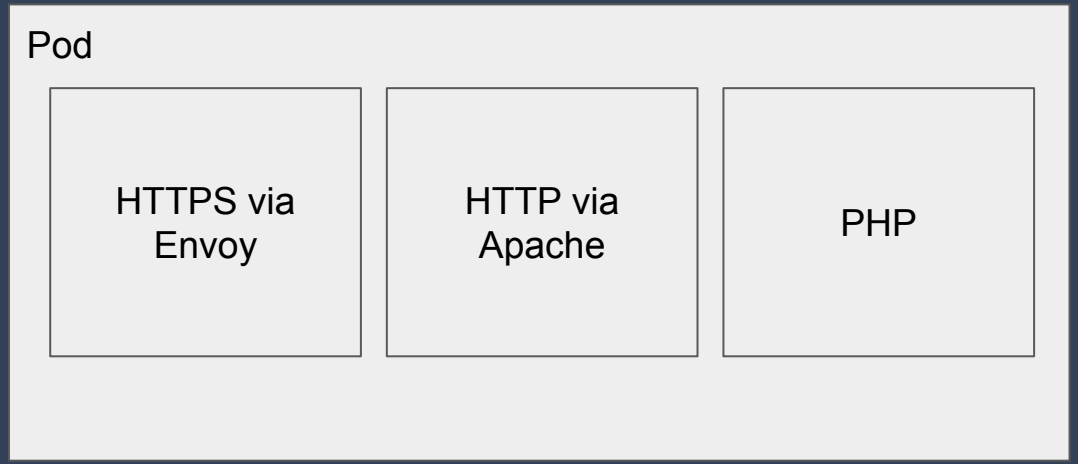
Example: a webserver and a HTTPS provider



Sidecar Pattern

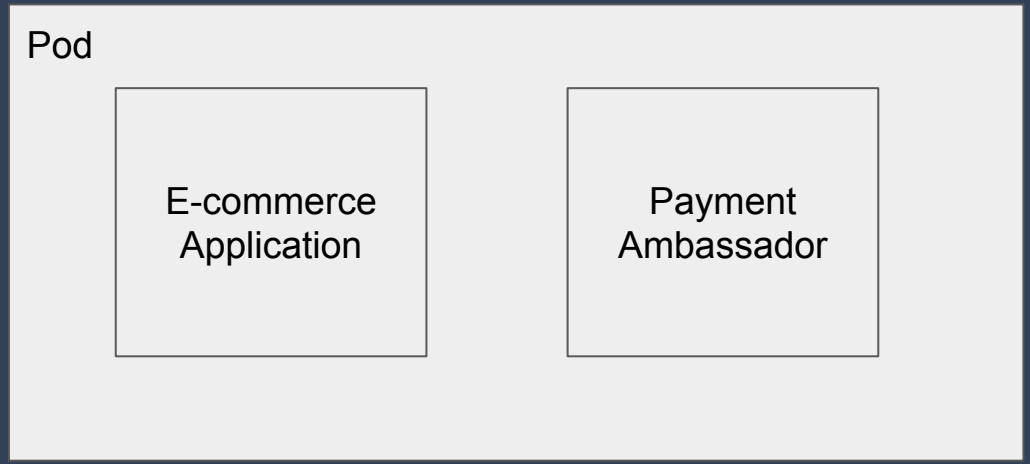
We can put more than one container in a pod, all of them will be scheduled and run concurrently and they share the network address space.

Example 2: an application server, a webserver and a HTTPS provider



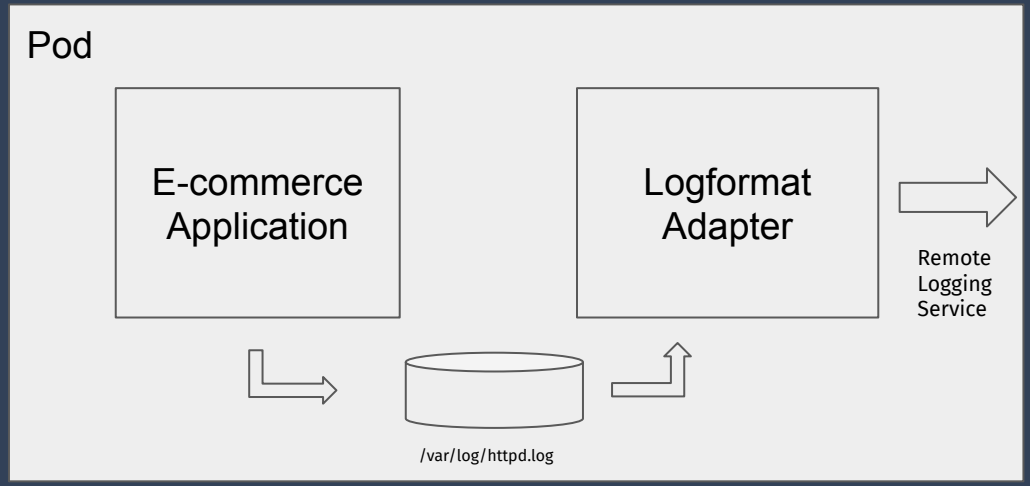
Ambassador Pattern

- Ambassador - a container that sends request on behalf of the application
  Example: a proxy to perform remote service requests API calls
  - Retry
  - Circuit-breaking
  - Security
  - Monitoring



Adapter Pattern

- Adapter - a container that reformats and enriches data for the main container
  Example: container reads web logs and reformat to JSON



Init Container

An Init Container will be started before the normal container(s) and needs to complete successfully before the other containers will be started. Imagine a case where you need to initialize and prepare a data structure that the other containers will use to serve requests.

```
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
spec:
  containers:
  - name: myapp-container
    image: busybox:1.28
    command: ['sh', '-c', 'echo The app is running! && sleep 3600']
  initContainers:
  - name: init-mydb
    image: busybox:1.28
    command: ['sh', '-c', "until nslookup mydb.default.svc.cluster.local; do echo waiting for mydb; sleep 2; done"]
```

Init Container

An Init Container will be started before the normal container(s) and needs to complete successfully before the other containers will be started. Imagine a case where you need to initialize and prepare a data structure that the other containers will use to serve requests.

```
wkandek:~/projects/k8s/ik/init$ kubectl apply -f d.yaml
pod/myapp-pod created
wkandek:~/projects/k8s/ik/init$ kubectl get -f d.yaml
NAME        READY   STATUS     RESTARTS   AGE
myapp-pod   0/1     Init:0/1   0          5s
wkandek:~/projects/k8s/ik/init$ kubectl apply -f  s.yaml
service/mydb created
wkandek:~/projects/k8s/ik/init$ kubectl get -f d.yaml
NAME        READY   STATUS     RESTARTS   AGE
myapp-pod   0/1     Init:0/1   0          22s
wkandek:~/projects/k8s/ik/init$ kubectl get -f d.yaml
NAME        READY   STATUS    RESTARTS   AGE
myapp-pod   1/1     Running   0          27s
```

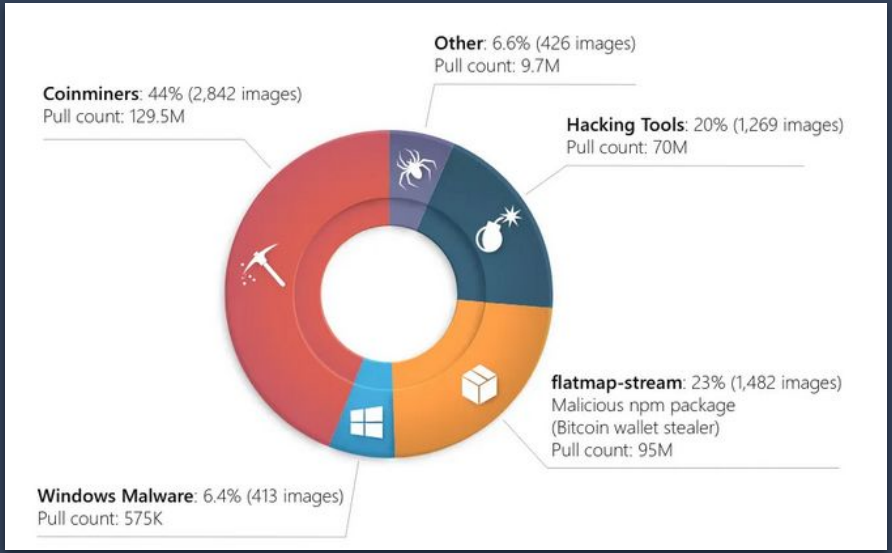
See: https://kubernetes.io/docs/concepts/workloads/pods/init-containers/

Kubernetes Security

- What images do you allow running?

Kubernetes Security

- What images do you allow running?



Kubernetes Security

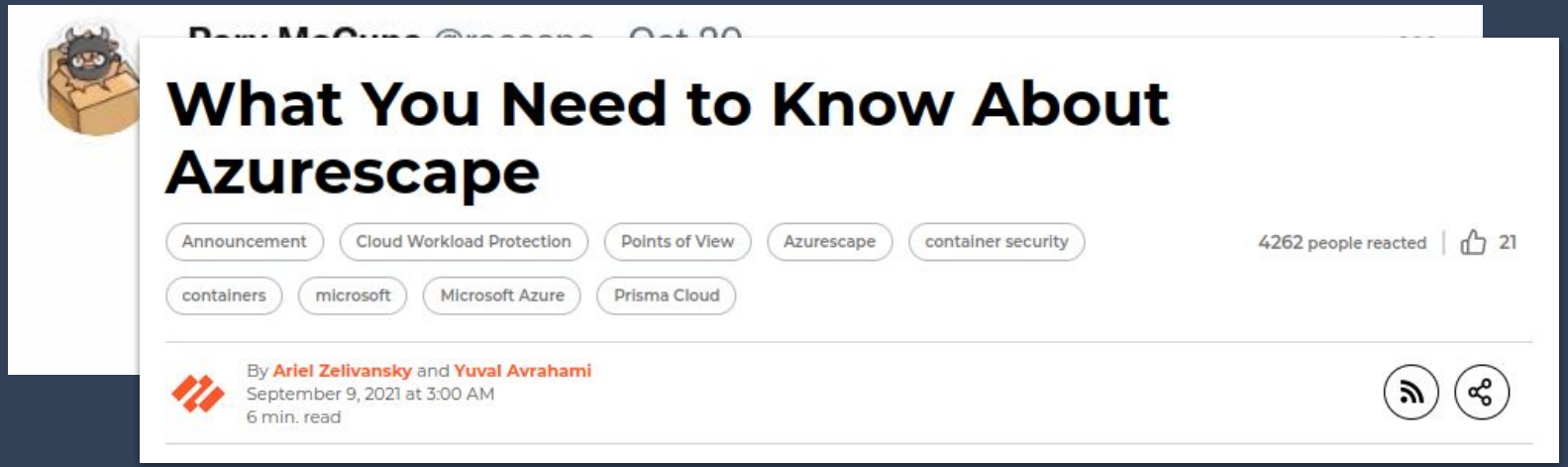
- What images do you allow running?
- Kubernetes is a fast moving project with quarterly releases and a one year support cycle

Kubernetes Security

- What images do you allow running?
- Kubernetes is a fast moving project with quarterly release and a one year support cycle

Kubernetes Security

- What images do you allow running?
- Kubernetes is a fast moving project with quarterly releases and a one year support cycle



Kubernetes Security

- What images do you allow running?
  - Control images
  - Update images frequently
  - Scan for problems
  - Minimize image content
- Kubernetes is a fast moving project with quarterly release and a one year support cycle
  - Update timely
- Kubernetes built-in Pod Security Standards/Admission
  - Privileged, baseline, restricted
    - Do not run containers as root = restricted
    - Read-only filesystem
  - Try to run as restricted
  - GKE has a limited runtime called gvisor
- Additional tools: logging, anomaly detection

Kubernetes Troubleshooting

- Does the container work under docker?
- kubectl top
- kubectl logs
- kubectl describe
- kubectl exec
- dashboard (minikube dashboard)
- Lens

- lsns/nsenter

Service Level Objectives

A Service Level Objective (SLO) is an understanding between teams about expectations for reliability and performance

- Our services will never have 100% reliability, but users don’t mind minor service interruptions
- Reliability expectations are often mismatched without an explicit agreement between teams
- Measuring from a **user level perspective** informs us on when to prioritize reliability work
- SLOs for each service and its dependent services tell us where to focus that work

- Same reasoning applies for performance

Service Level Objectives

We are talking about Product reliability and SLOs give us a common ground

- Product
    - SLOs in product planning gives reliability a seat at the feature table
- Development
    - SLOs influence implementation choices and incentivize metric generation
- Operations
    - SLOs give operations clarity on operating priorities and parameters

Service Level Objectives Concepts

- Service Level Objective (SLO)
  - A goal or cutoff using a measured value
    - More than 99% of all request are successful
    - Less than 0.5% of pages are outdated

Service Level Objectives Concepts

- Service Level Objective (SLO)
  - A goal or cutoff using a measured value
    - More than 99% of all request are successful
    - Less than 0.5% of pages are outdated

- Service Level Indicator (SLI)
  - A measured value or metric, best expressed in %
    - Ratio of requests that are successful
    - Ratio of outdated pages served

Service Level Objectives Concepts

- Error Budget
  - An SLO implies an acceptable level of unreliability
    - This is a budget that can be allocated
    - Example: fast burn - slow burn

Service Level Indicators

$$\text{SLI}: \left(\frac{\text{good events}}{\text{all events}} \right) * 100\%$$

- Intuitive
  - 100% is good
  - 0% is bad
- Consistent
  - SLI/O practitioners will recognize the values and that makes it easier to talk and reason about

Service Level Indicators

- Request/Response
  - Latency
  - Availability
  - Quality
- Storage
  - Durability
- Data Processing
  - Freshness
  - Coverage
  - Correctness
  - Throughput

Service Level Indicator Example

- Availability - The orders page should load successfully
  - Define “success”
  - Where is this recorded?
- The proportion of valid requests served successfully =>
- The proportion of HTTP GET requests for /orders served successfully: return code 2XX, 3XX, measured at the web server

- Latency - The oreders page should load quickly
  - Define “quickly”
  - How does you time this?
- The proportion of valid requests served faster than the threshold =>
- The proportion of HTTP GET requests for /orders served faster than 250 ms, measured at the web server

Service Level Objective Reporting

Your objectives should have both a target and a measurement window

Service	SLO Type	Objective
Web: Orders	Availability	99.95% successful in previous 28 days
Web: Orders	Latency	90% of requests < 500ms in previous 28 days
...	...	...

Interview Question Areas

- Docker build, layers, size minimization
- Kubernetes Components (control plane, nodes) - who does what
  - Declarative System, API, etcd, controllers
- Kubernetes Liveness, Readiness and S?????? probes
- Kubernetes Pods vs containers (sidecars)
- Kubernetes troubleshooting

Resources and Q&A

- Google SRE Books - https://sre.google/books/
- Kubernetes - https://kubernetes.io
- Architecture - https://kubernetes.io/docs/concepts/architecture
- Components - https://kubernetes.io/docs/concepts/overview/components
- Networking - https://kubernetes.io/docs/concepts/cluster-administration/networking/

  Demo doc: https://docs.google.com/document/d/1anjaK5f0MJ5fc0OpgyDV1EjvolPGFH-P9lEkcU8Z3BA/edit
- Storage - https://kubernetes.io/docs/concepts/storage
- K8s the hard way - https://github.com/kelseyhightower/kubernetes-the-hard-way
- nsenter example - https://yoheimuta.medium.com/debugging-gke-unprivileged-containers-with-gdb-and-nsenter-3760b50eb03a
- SLO Dashboard at GitLab - https://dashboards.gitlab.com
- SLO Class at Coursera - https://www.coursera.org/learn/site-reliability-engineering-slos