HI, I’M HEIKO

- I’m a Senior Consultant from a company you never heard
- I help putting EVs in the Cloud
- (Event) Photography
- Longboarding

/in/heikoborchers/
@archang3l_media
@archang3l_media@chaos.social
ABOUT ME

HI, I’M CEDI

I’m a Senior SRE at $BigTech

I do resiliency engineering

(Analog) Photography

Brazilian Jiu-Jitsu

/cedi /in/cekienzl

@cedi@hachyderm.io
CONTAINERS, CONTAINERS EVERYWHERE

CONTAINER & VM
PRo

• Consistency
• Automation
• Stability
• Scalability

COCON

• Yet another layer...
• Learning Curve
• Documentation
• Plethora of tools
CONTAINER! WHY?

VM ORCHESTRATION
CONTAINER! WHY?

CONTAINER ORCHESTRATION

- Octopus
- Docker
- Kubernetes
ORCHESTRATION

CONTAINER
CONTAINER ORCHESTRATION

DOCKER COMPOSE
COMPOSE is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services. Then, with a single command, you create and start all the services from your configuration.
CONTAINER ORCHESTRATION

DOCKER-COMPOSE

File: docker-compose_clean.yml

version: "3.5"

services:
  gitlab:
    image: 'gitlab/gitlab-ce:latest'
    restart: always
    hostname: 'gitlab.example.com'
    environment:
      GITLAB_OMNIBUS_CONFIG:
        external_url: 'http://gitlab.example.com'
        registry_external_url: 'https://registry.gitlab.example.com'
        gitlab_rails[gitlab_shell_ssh_port] = 2224
        gitlab_rails[smtp_enable] = "true"
        gitlab_rails[smtp_address] = "some-mail-server.example.com"
        gitlab_rails[smtp_port] = "587"
        gitlab_rails[smtp_user_name] = "noreply@example.com"
        gitlab_rails[smtp_password] = "example_password"
        gitlab_rails[smtp_domain] = "example.com"
        gitlab_rails[smtp_authentication] = ""'
        gitlab_rails[smtp_tls] = "true"
        gitlab_rails[registry_enabled] = "true"
        gitlab_rails[registry_host] = "registry.gitlab.example.com"
        gitlab_rails[registry_port] = "50000"
        gitlab_rails[gitlab_default_projects_features_container_registry] = "true"
        gitlab_rails[registry_path] = "/opt/storage/gitlab/gitlab-rails/shared/registry"
        nginx[real_ip_header] = "X-Real-IP"
        nginx[real_ip_recursive] = "on"
        nginx[listen_http] = false
        nginx[listen_port] = 80
        registry_nginx[ssl_certificate] = "/gitlab/certs/fullchain.pem"
        registry_nginx[ssl_certificate_key] = "/gitlab/certs/privkey.pem"

ports:
  - "$0001:80"
  - "$3306:3306"
  - "$5000:5000"

volumes:
  - "/opt/storage/gitlab/config//etc/gitlab"
  - "/opt/storage/gitlab/logs:/var/log/gitlab"
  - "/opt/storage/gitlab/data:/var/opt/gitlab"
  - "/gitlab/certs:/gitlab/certs"
PRO

- Easy
- Multiple Environments on a single host
- Stability
- Reproducible
- Declarative / Infrastructure as Code

CON

- Single Host Deployments
CONTAINER ORCHESTRATION

CLOUD SOLUTIONS

Azure

DigitalOcean

aws

Magenta CLOUD
YES, BUT...
HONORABLE MENTIONS

CONTAINER ORCHESTRATION

HONORABLE MENTIONS
NOW, WHAT _IS_ KUBERNETES?

SPOILER ALERT,
IT’S NOT A SINGLE TOOL
KUBERNETES DEFINES A SET OF BUILDING BLOCKS ("PRIMITIVES") THAT COLLECTIVELY PROVIDE MECHANISMS THAT DEPLOY, MAINTAIN, AND SCALE APPLICATIONS
DISTRIBUTIONS

KUBERNETES DISTRIBUTIONS
DISTRIBUTIONS

DEVELOPMENT

SINGLE NODE / EDGE

PRODUCTION

KUBERNETES

MicroK8s

kind

OSID

K8S

kops
DISTRIBUTIONS... CONTINUED...
BUT WHEN?

KUBERNETES
KUBERNETES, BUT WHEN?

LET'S PLAY A GAME....

Do you want to play a game?
IS KUBERNETES THE RIGHT CHOICE FOR SERVING STATIC CONTENT?
IS KUBERNETES THE RIGHT CHOICE FOR DATABASES?
NO!!!
IS KUBERNETES THE RIGHT CHOICE FOR

PLANT / FACTORY / PHYSICAL SAFETY
CONTROL SOFTWARE?
NO!!!
IS KUBERNETES THE RIGHT CHOICE FOR STATEFUL WORKLOAD?
IT DEPENDS
NO* 
* when state is kept in the software
YES*

* when state is kept external (in a database cluster or object storage)
IS KUBERNETES THE RIGHT CHOICE FOR YOUR COMPANIES (LEGACY/ENTERPRISE) JAVA APPLICATION?
MAYBE?
**PRO**

- Improve density (in comparison to running each Application on a single host)
- Isolation (no JVM dependency chaos)
- State is probably (hopefully) stored somewhere else anyway

**CON**

- It will not scale (better) than before
- Low utilization of Kubernetes native features
IS KUBERNETES THE RIGHT CHOICE FOR EDGE COMPUTING?
MAYBE?
**PRO**

- Running your services in a standardized way close to the consumer
- Distributed sync
- State can (and should) be stored centrally

**CON**

- Managing multiple Kubernetes Clusters requires much more planning
- Multi-Cluster deployments are hard to facilitate
- Latency constraints for inter-cluster communication
STATELESS MICRO-SERVICES?

IS KUBERNETES THE RIGHT CHOICE FOR
YES!
IS KUBERNETES THE RIGHT CHOICE FOR SERVERLESS COMPUTING?
YES*!  * but you need additional software
DEFAULT TO KUBERNETES ONLY WHEN THERE’S NOT A BETTER OPTION FOR YOUR WORKLOADS.
IS KUBERNETES REALLY THE RIGHT CHOICE FOR YOU?

▸ What are your functional infrastructure requirements?
▸ What are your scaling requirements?
▸ How do you handle state in your workload?
▸ Do you have the operational capacity to operate a Kubernetes cluster?
EVALUATE CAREFULLY

POSSIBLE ALTERNATIVES

▸ IaaS: Hetzner, Azure VMSS, AWS EC2
▸ PaaS: Heroku, Azure WebApps, AWS Elastic Beanstalk
▸ CaaS: DigitalOcean Droplets, Azure Container Instances, AWS Fargate
▸ FaaS: Azure Functions, AWS Lambda
KUBERNETES

BUT HOW?
GET STARTED WITH A DEV ENVIRONMENT

- Make your first steps with “kind” on your local machine
BUT HOW (TO GET STARTED)

SINGLE NODE-, EDGE-, (AND HOMELAB) DEPLOYMENT

- K3s on a single node is good enough for your home-lab
- Multi-node K3s is probably enough for most use-cases!
- Use a systemd service unit to keep k3s running
- Use the K3s Ansible Playbook
BUT HOW (TO GET STARTED)

IF YOU REALLY WANT TO GO ALL THE WAY

- Use the Cluster API
  - Declarative Management isn’t just nice for workload but for entire clusters too
- Read the documentation of clustermgr
- Since most of you deploy to hetzner anyway
  - ccl.pw/cluster-api-hetzner
- This isn’t gonna be easy! But we warned you :)}
DEPLOY WORKLOAD
DEPLOY A WORKLOAD

PODS

POD ➔ CONTAINER
MANAGING PODS

Deploy a Workload

Deployment ➔ Replica Set ➔ Pod ➔ Pod
MANAGING PODS

(CORN) JOBS

DAEMON SET

STATEFUL SET

POD

RUN A POD ONCE/RECURRING

RUN ONE POD ON EACH VM

DO MAGIC WITH THE POD
A WORD ON “KUBECTL APPLY -F” AND “HELM INSTALL”

PLEASE DON’T*

*I mean, it’s fine on dev. But certainly not on prod
DON’T INSTALL ANYTHING MANUALLY ON YOUR (PROD) CLUSTER

- Not reproducible
- Leads to “Snowflake” Deployments
- Hard to audit (what, when, and why?)
- Keeping a historical record of changes is hard
- Almost impossible to do proper secrets management
BUT HOW?

GIT-OPS TO THE RESCUE
DEVELOPMENT

IDE

Continuous Integration

Test

GIT

DEPLOYMENT

Deployment (clusters, apps)

Management (operations)

Monitoring Tracing Logging (Observability)

Kubernetes GitOps
**PUSH**

- Continuous Delivery Pipeline applies configuration to Kubernetes Cluster
- Requires your CD Pipeline to have access to the Cluster
- Does not detect configuration drift

**PULL**

- GitOps Operator runs inside the Kubernetes Cluster and pulls changes from Git and applies them to the Cluster from the inside
- Requires the GitOps Operator to have access to the Repository
- Can detect configuration drift and revert manual changes
ARGO-CD

- Easy to use
- Declarative
- Nice looking management UI
- Easily extensible with your favorite secret management solution
- Avoid config drift
  - Reconcile loop can roll-back manual changes automatically
GIT-OPS
SECRET MANAGEMENT
SECRET MANAGEMENT

GIT-CRYPT
- Encrypts whole files
- Based on PGP
- Git-Diff won’t work at all

MOZILLA SOPS
- Encrypts only YAML Value
- Can use a multitude of credential providers and encryption methods (PGP, AGE, Vault, ...)
- Git-diff works (to some extent)
HOW TO SERVE YOUR APPLICATION
Pod-Spec exposes a Port

A Service selects Pods via labels

Service specifies an exposed Port

The Service ensures traffic is served to the Pod on the Pods Port
HOW TO SERVE CUSTOMER TRAFFIC

- Node Port
- Load Balancer
- Ingress Controller
EASY MODE: NODE PORT

- No Load-Balancing and auto-failover
- Available Node-Port Range is 30000-32767
- No reserved ports (80, 443, etc.)
**NETWORKING**

**NODE-PORT**

Diagram showing the communication flow between a client and a service, with a pod involved. The code snippets illustrate the Kubernetes `Deployment` and `Service` definitions for a Kubernetes cluster, demonstrating how services are deployed and accessed through node ports.
**ADVANCED MODE: LOADBALANCER**

- Expensive (usually each LoadBalancer is billed separately, for existence & traffic)
- Certificate management is limited by the Cloud-Provider capabilities
- Requires a Cloud-Provider with LoadBalancer support
- TLS usually terminated at the LB
- Only works for domains, not paths
NETWORKING

LOADBALANCER

CLIENT  →  LB  →  SERVICE  →  POD

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kuard-deployment
labels:
  app: kuard
spec:
  replicas: 3
  selector:
    matchLabels:
      app: kuard
template:
  metadata:
    labels:
      app: kuard
  spec:
    containers:
      - image: gcr.io/kuar-demo/kuard-amd64:blue
        name: kuard
        ports:
          - containerPort: 8080
            name: http

apiVersion: v1
kind: Service
metadata:
  name: kuard-service
spec:
  selector:
    app: kuard
  ports:
    - port: 80
      targetPort: 8080
```
99.9% of the time simply use ingress-nginx or envoy

Domain Auto discovery

Certificates can be issued automatically

High performance

Works on URL paths
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tailnet-ingress
  annotations:
    ingress.kubernetes.io/rewrite-target: /
    kubernetes.io/ingress.class: nginx
cert-manager.io/cluster-issuer: "letsencrypt-prod"
spec:
rules:
- host: test.cedi.dev
  http:
    paths:
    - path: /
      pathType: Prefix
      backend:
        service:
          name: kuard-service
          port:
            number: 80
tls:
  - secretName: test-ts-secret
    hosts:
    - test.cedi.dev

apiVersion: v1
kind: Service
metadata:
  name: kuard-service
spec:
  selector:
    app: kuard
  ports:
  - port: 80
    targetPort: 8080

apiVersion: apps/v1
kind: Deployment
metadata:
  name: kuard-deployment
  labels:
    app: kuard
spec:
  replicas: 3
  selector:
    matchLabels:
      app: kuard
  template:
    metadata:
      labels:
        app: kuard
    spec:
      containers:
      - image: gcr.io/kuard-demo/kuard-amd64:blue
        name: kuard
        ports:
        - containerPort: 8080
          name: http

CLIENT → LB → INGRESS → SERVICE → POD
KUBERNETES BUT HOW

WRAP-UP

- Take a Pi, install K3s
- Deploy your workload with Helm and or Kustomize
- Secure your secrets with SOPS and don’t leak them to Git
- Keep your Cluster reconciled with ArgoCD
- Use ingress-nginx to serve incoming requests

CONGRATS, NOW YOU HAVE A K8S CLUSTER TO MAINTAIN
THANKS FOR LISTENING!

Q&A TIME
SHAMELESS SELF-PLUG

RELATED TALKS AND TOPICS

- Decoding Site Reliability Engineering: An Exploration of SRE, DevOps, and Platform Engineering
  - [ccl.pw/decoding-sre](ccl.pw/decoding-sre)
- Modern Observability - Scalable Observability with the LGTM Stack: Harnessing the Power of Loki, Grafana, Tempo, and Mimir
  - [ccl.pw/modern-o11y](ccl.pw/modern-o11y)
- Understanding Alerting - How to come up with a good enough alerting strategy (GPN 20)
  - [ccl.pw/alerting](ccl.pw/alerting)
- Kubernetes, the good, the bad and the Ugly (GPN 20)
  - [ccl.pw/k8s-the-good-the-bad-the-ugly](ccl.pw/k8s-the-good-the-bad-the-ugly)