QA-Deployment with Kubernetes

How to deploy multiple QA environments with the help of K8s

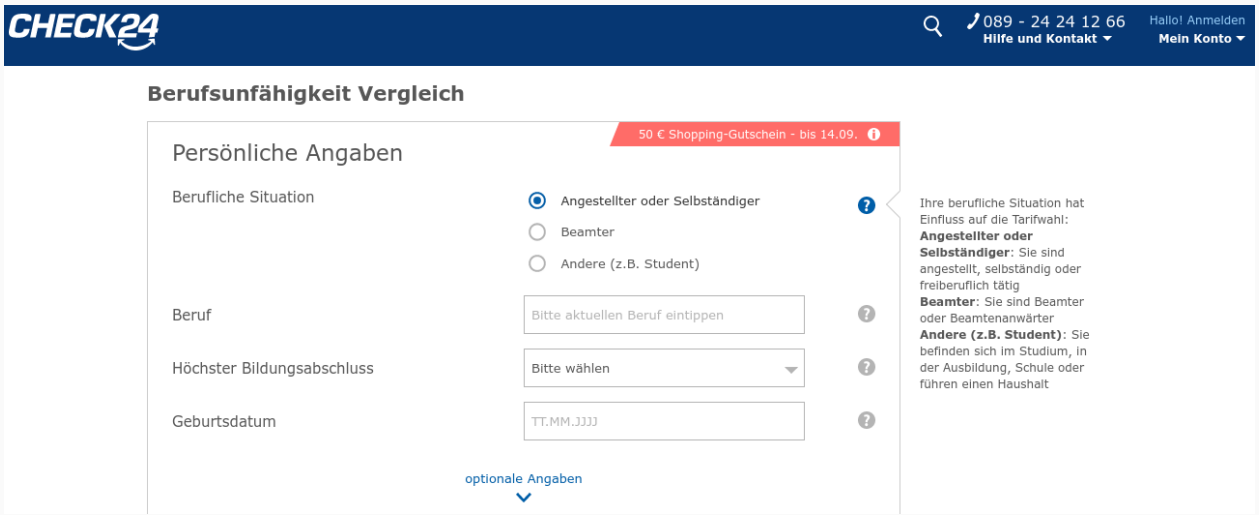
Jan Baer, LeadDeveloper

24. Februar 2021

CHECK24

BU-Product overview

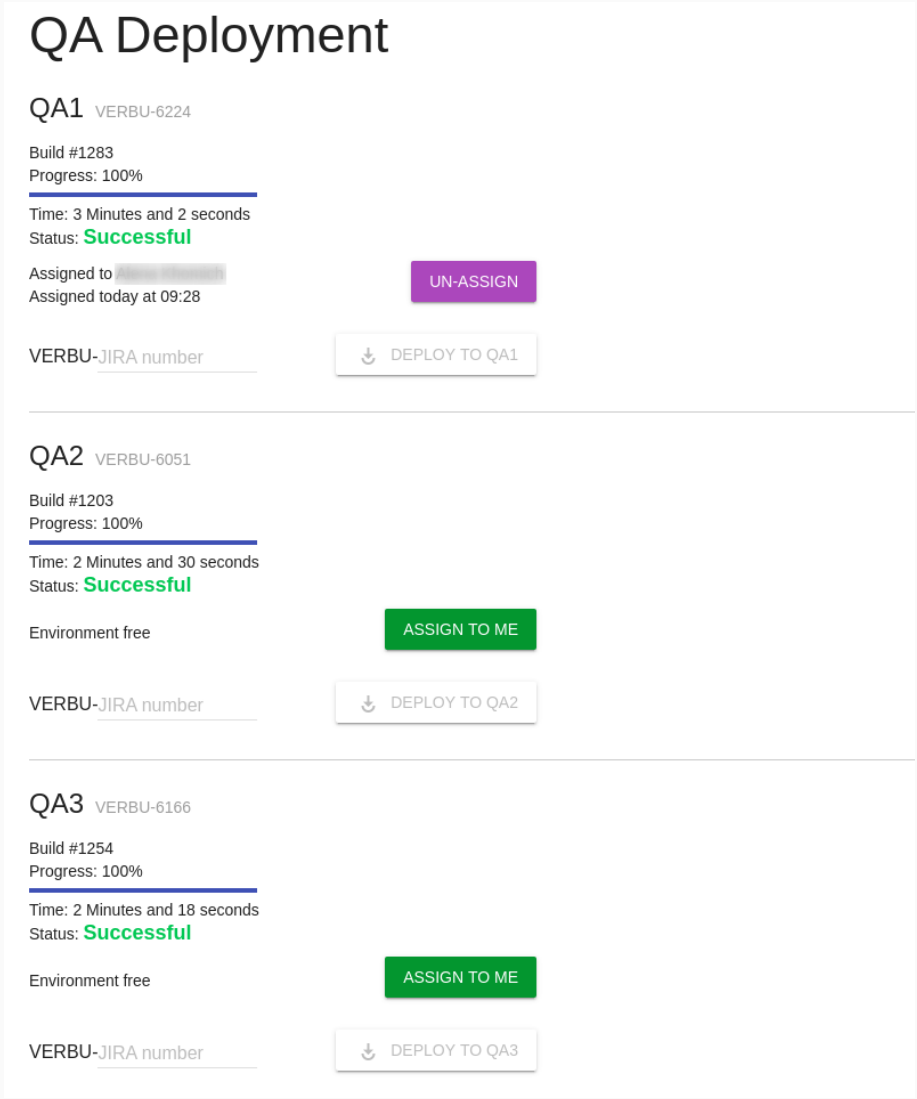
- Comparison for disability insurances - 2 different products, separate desktop and mobile apps
- Backoffice apps
- Administration apps
- Node.js, Next.js, Docker, Docker-compose
- 6 Developers, 2 QA-Engineers, 4 Productmanagers



What we had before

- 3 QA environments on 3 VMs
- 3 cloned Bamboo plans for deployment
- 3 configuration files for each project with fixed host-urls (qa1, qa2, qa3)
- Docker-compose with 68 running containers on each host machine (In the meantime we have 74)
- HAProxy for routing (no loadbalancing)
- Very difficult to add more QA environments (new VM, build-plan etc.)
- Hard to investigate when a feature was not deployed correctly

How the UI was looking



What was on our wishlist

- At least double the QA environments
- Easier scalable if necessary
- Only one config for all QA environments
- Better management and error investigation
- Stable URLs per feature-deployment

What options we had to improve that

- Create more VMs and setting up new Bamboo plans
- Improving speed for creation of VMs with using a more automated way for bootstrap (Terraform)
- Reduce number of containers to deploy with building groups to test

How about using Kubernetes

What features from Kubernetes could help us

- Isolation between parallel deployments with using namespaces
- Dynamic generation of Urls with Ingress controller
- Restful API for deploying from external service
- Scalable cluster architecture

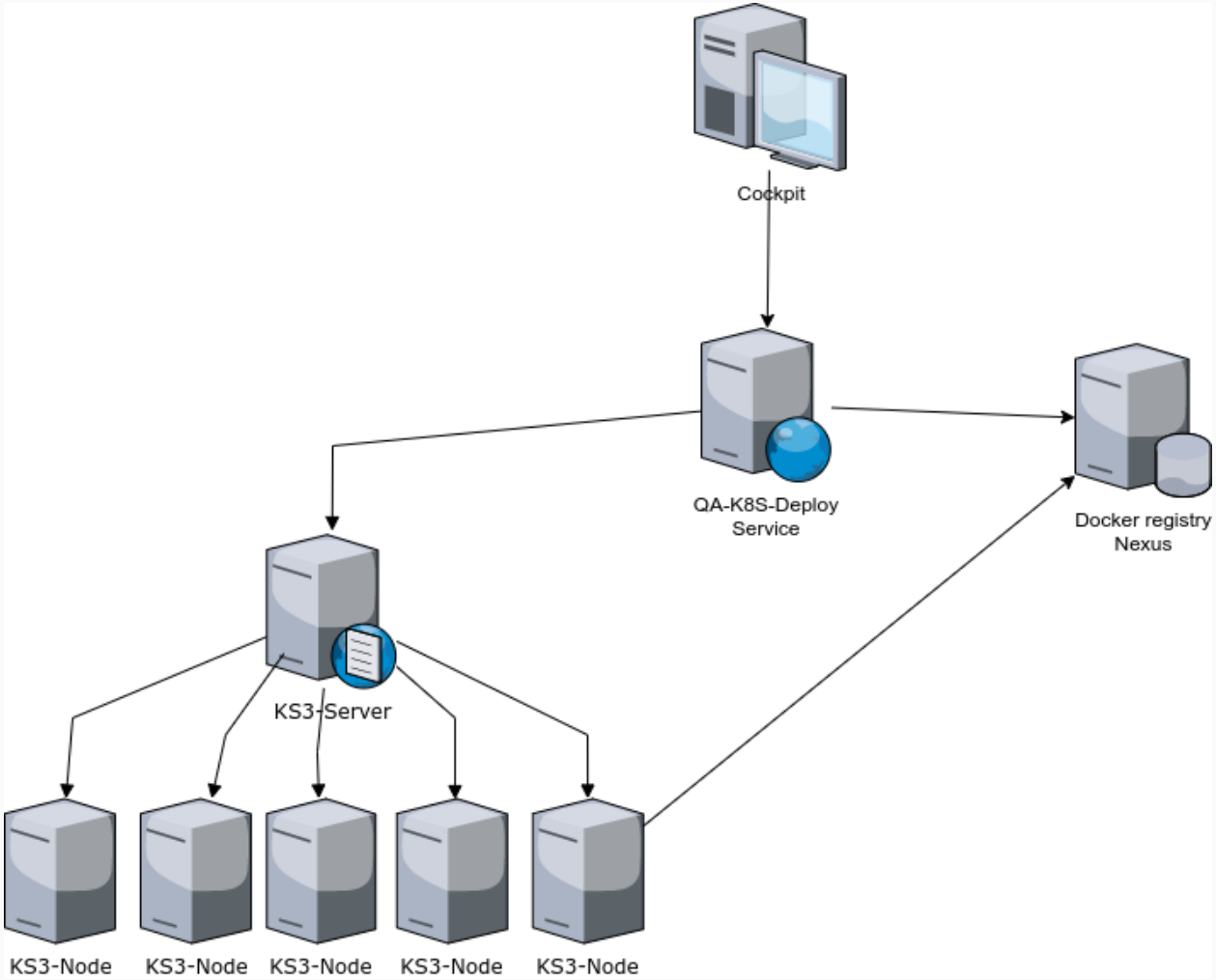
What comes from us

- Cockpit - provides a lot of functionalities for our daily workflows with Testing and Deployment
- QA-K8S-Service - Microservice with endpoints for creating, updating, and deleting qa-deployments

What external components we use

- Nexus Docker Registry
- K3s - Lightweight Kubernetes Distribution
- Rancher - Kubernetes management platform

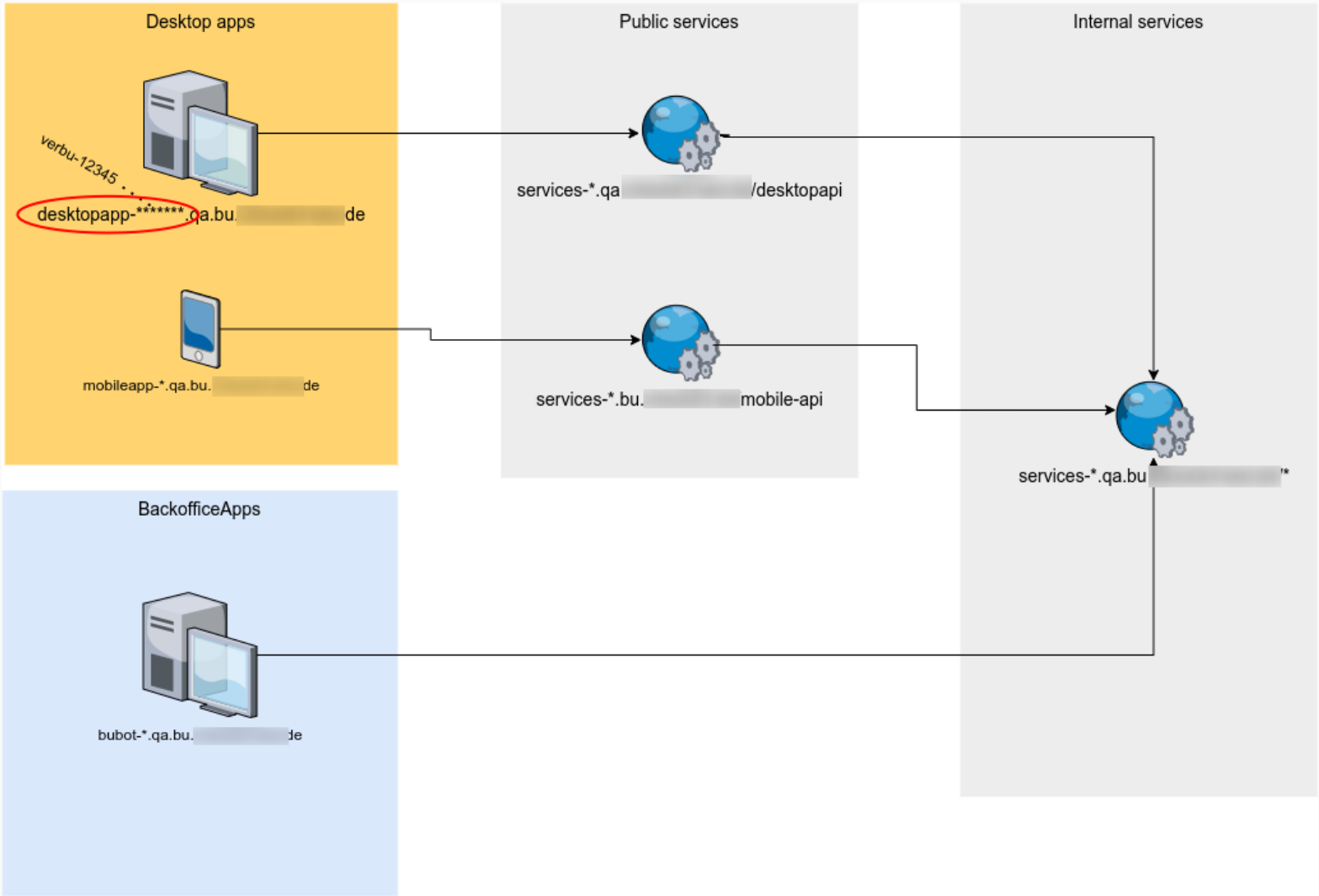
How is it working together



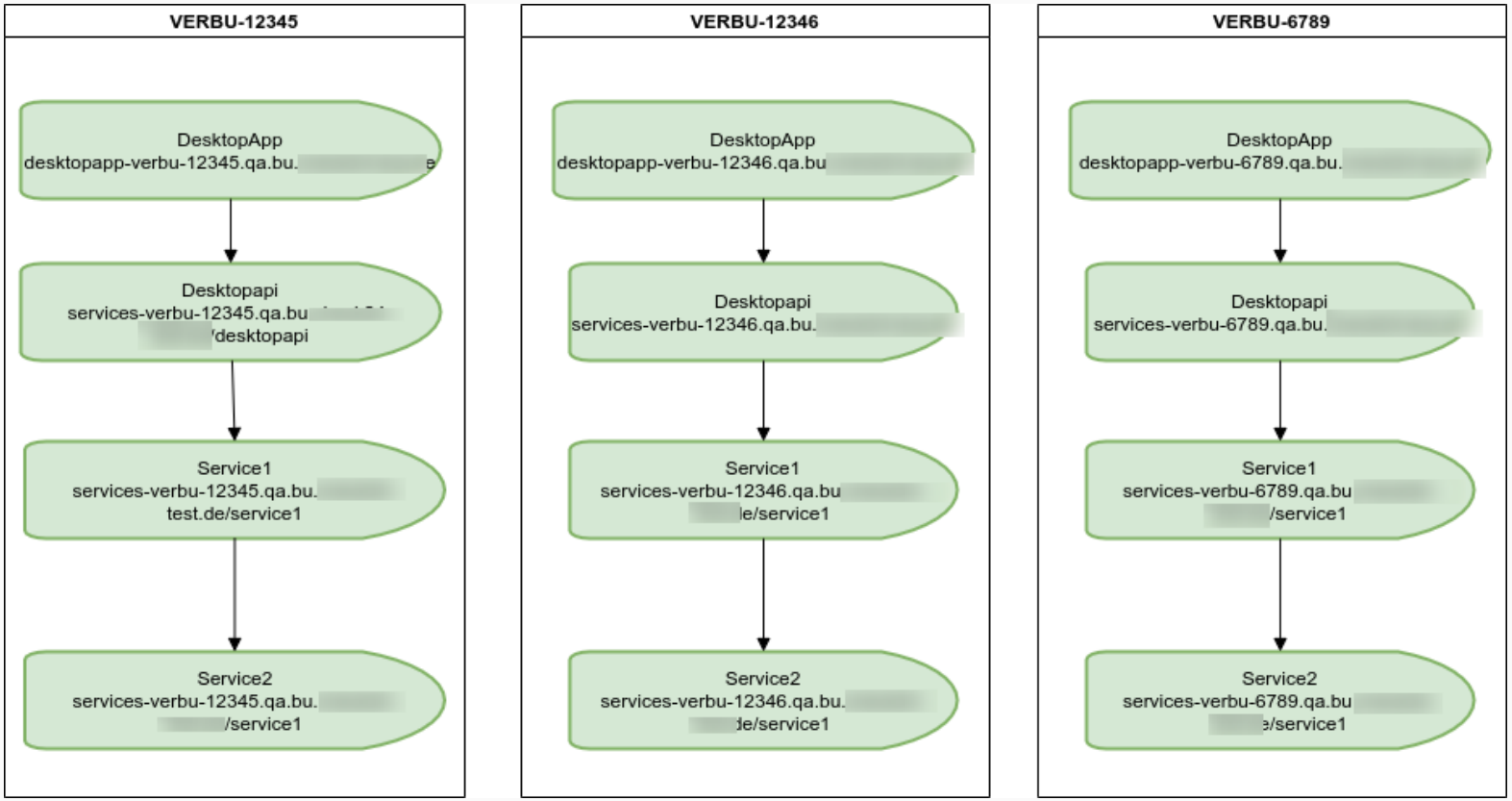
How to communicate with Kubernetes

- Using officially-supported Kubernetes client libraries - Link
- Using REST api directly - Link
  - Tip: run `kubectl ... -v 8` to see the rest requests for each command

How services and apps are communicating with each other



How to deploy services and apps to QA (v1)



What problems we had to solve

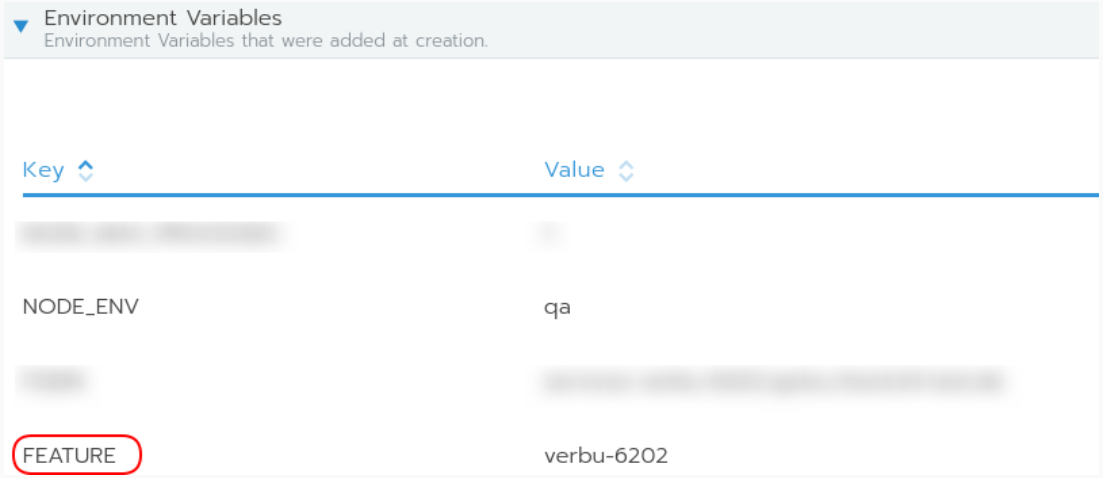
- How to authorize to talk with Kubernetes
- Dynamic creation of urls
- Waiting for depending services (NSQ)
- Find the right limits
- Rewriting urls
- Updating deployments

How to authorize to talk with Kubernetes

- Create service-account for **qa-deploy**
- Assign role to be able to create or delete namespaces and deploy PODs (RBAC)
- How to talk with Kubernetes over *https* (use server_ca.crt)

Dynamic creation of urls

Set feature as environment variable



- Use placeholders in config files
- Replace placeholder with feature env when service is starting

```
{
  "env": "qa",
  "origin": "https://services-${feature}.qa.bu.
  "services": {
    "auth": "https://services-${feature}.qa.bu.
    "ripple": "https://services-${feature}.qa.bu.
    "miami": "https://services-${feature}.qa.bu.
    "customerActivities": "https://services-${feature}.qa.bu.
  },
```

Waiting for dependent services

- Some of the services require a running NSQ service

```
<% if ( requiresNsq ) { %>
initContainers:
  - name: wait-for-nsq
    image: subfuzion/netcat
    command: ['sh', '-c', "while ! nc -z nsqd 4151; do sleep 0.5; done"]
<% } %>
```

State	Name	Image	Restarts
Waiting PodInitializing	brain	[redacted]/docker/brain:master_latest	0
Waiting PodInitializing	wait-for-nsq Init Container	subfuzion/netcat	

Finding the right limits

- Observe a deployment to learn what resources are required

```
† ~
➜ watch -n 2 -t kubectl top pods -n verbu-6202
```

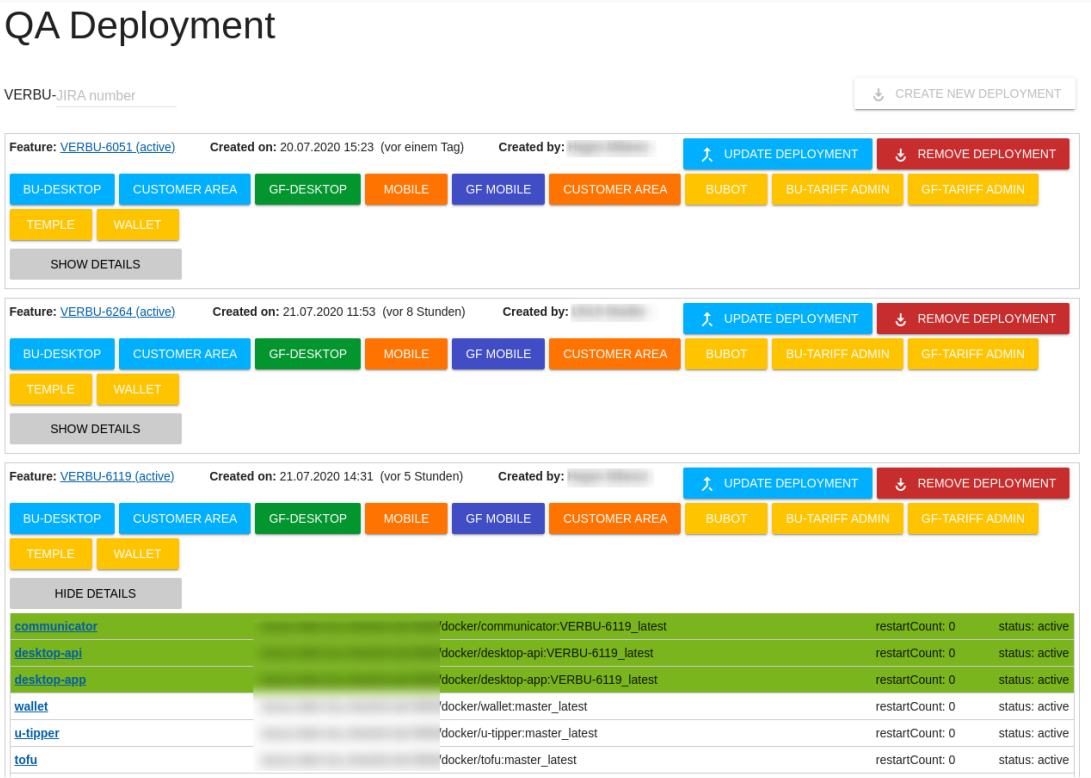
```
NAME                                                CPU(cores)   MEMORY(bytes)
accounting-api-5588fc9678-pd57h                     2m           65Mi
acid-6c84c7dff-cdm52                                1m           63Mi
addressservice-5b5bf7f4c7-4npfw                     1m           97Mi
auth-cb4c45b5b-522f5                                9m           62Mi
auth-ui-6cccc7b664-kwgq5                            1m           80Mi
brain-5c85f89b47-l6hlw                              2m           64Mi
bu-cleanup-fd45f8b85-4×75f                          1m           68Mi
bubot-748c6b76cb-tpwqx                              12m          110Mi
bubot-accounting-55bf8c85cf-7fc9w                   1m           70Mi
bubot-appointments-6796479ccb-khst9                 1m           93Mi
bubot-consulting-process-6646dd9665-t9fzp           1m           88Mi
bubot-documents-5fcd6f856-ksc5g                     24m          105Mi
bubot-insurance-application-requests-66b787796b-97rzd  1m        96Mi
bubot-mailing-5666f7dcc5-vvmxk                      1m           88Mi
bubot-rivo-586dd8f69b-zgxx7                         1m           71Mi
bubot-salary-6cf7f5d85f-lh929                       1m           67Mi
bus-5b86d84798-8xpfg                                1m           49Mi
coachman-94d74db9c-vd4sz                            22m          76Mi
communicator-7cd7d4968b-zk7rp                       1m           74Mi
consulting-process-api-549fcb8986-r8zzb             5m           66Mi
```

Rewriting of urls

- Remove `/eventbus` from the url before forwarding to NSQ service

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    kubernetes.io/ingress.class: traefik
    traefik.frontend.rule.type: PathPrefixStrip
  name: ingress-eventbus
  namespace: <%= namespace %>
spec:
  rules:
  - host: <%= host %>
    http:
      paths:
      - backend:
          serviceName: <%= name %>
          servicePort: <%= port %>
        path: <%= path %>
  tls:
    - hosts:
      - <%= host %>
      secretName: qa-bu-ssl-certificate
```

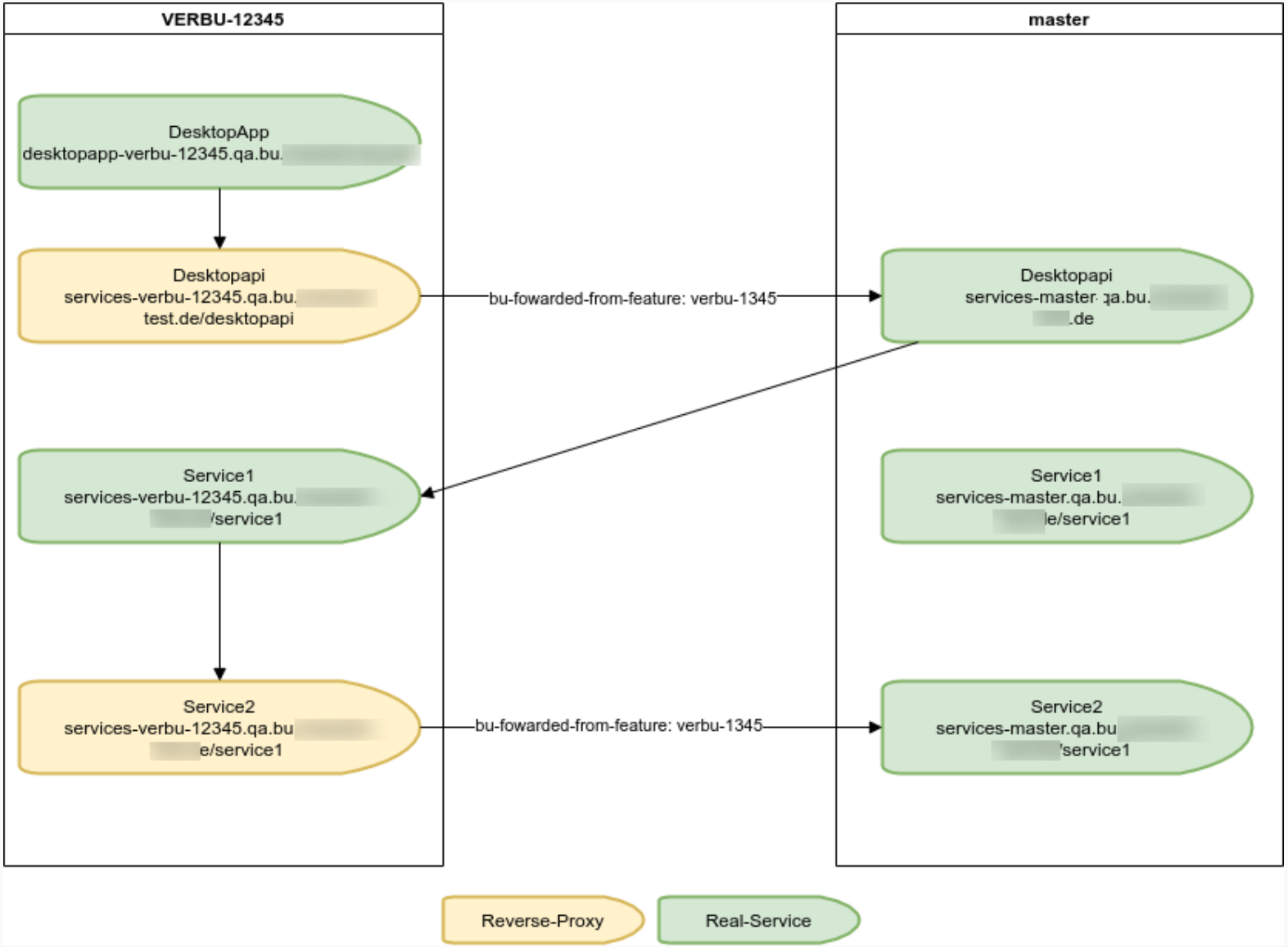
Mission completed?



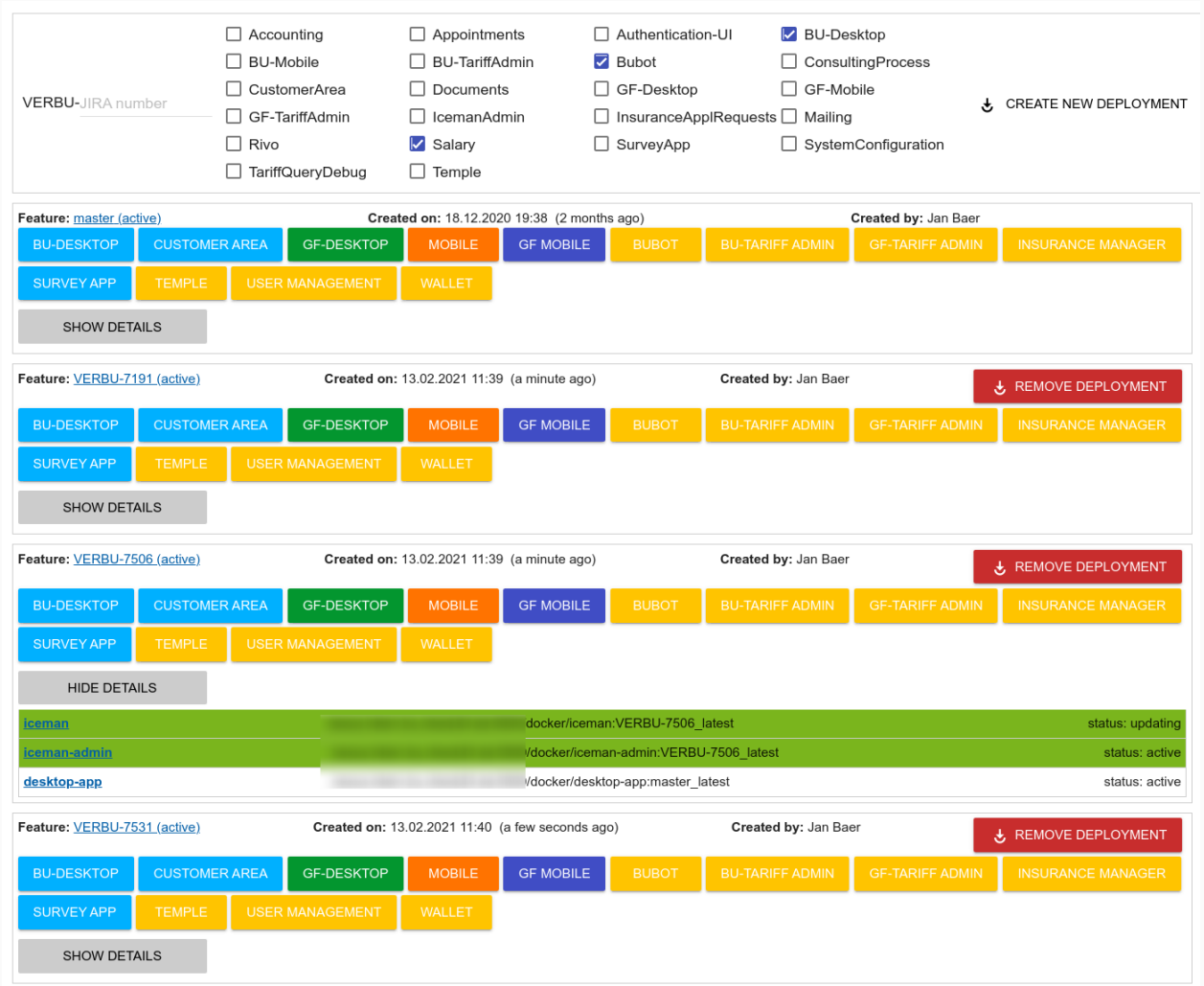
- Now we could deploy 5 parallel deployments (the 6th became instable)
- But each deployment requires ~70 PODs to deploy
- Creation but also deletion was slow
- Updating was faster but manual trigger required

How about using a ServiceMesh, like Istio?

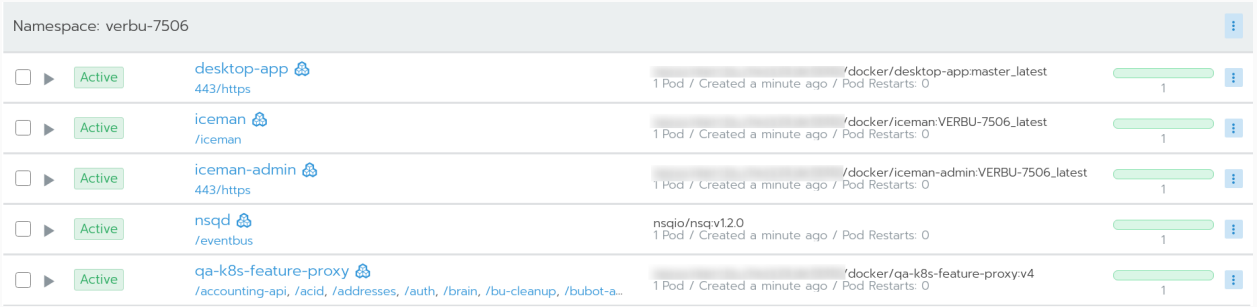
How to deploy services and apps to QA (v2)



How is it looking now?



What is running internally



How is it improving the deployment?

- Much faster deployment because of deploying only a few services and apps
- Faster cleanup of existing deployments
- Using much less resources per deployment
- More parallel deployments are possible
- Bonus: Automatic updating of deployments from Bamboo

What is K3s?

K3s is a fully compliant Kubernetes distribution with the following enhancements:

- Packaged as a single binary. (less than 100 MB.)
- Lightweight storage backend based on sqlite3 as the default storage mechanism. etcd3, MySQL, Postgres also still available.
- Wrapped in simple launcher that handles a lot of the complexity of TLS and options.
- Secure by default with reasonable defaults for lightweight environments.

What is K3s?

- Simple but powerful “batteries-included” features have been added, such as: a local storage provider, a service load balancer, a Helm controller, and the Traefik ingress controller.
- Operation of all Kubernetes control plane components is encapsulated in a single binary and process. This allows K3s to automate and manage complex cluster operations like distributing certificates.
- External dependencies have been minimized (just a modern kernel and cgroup mounts needed).

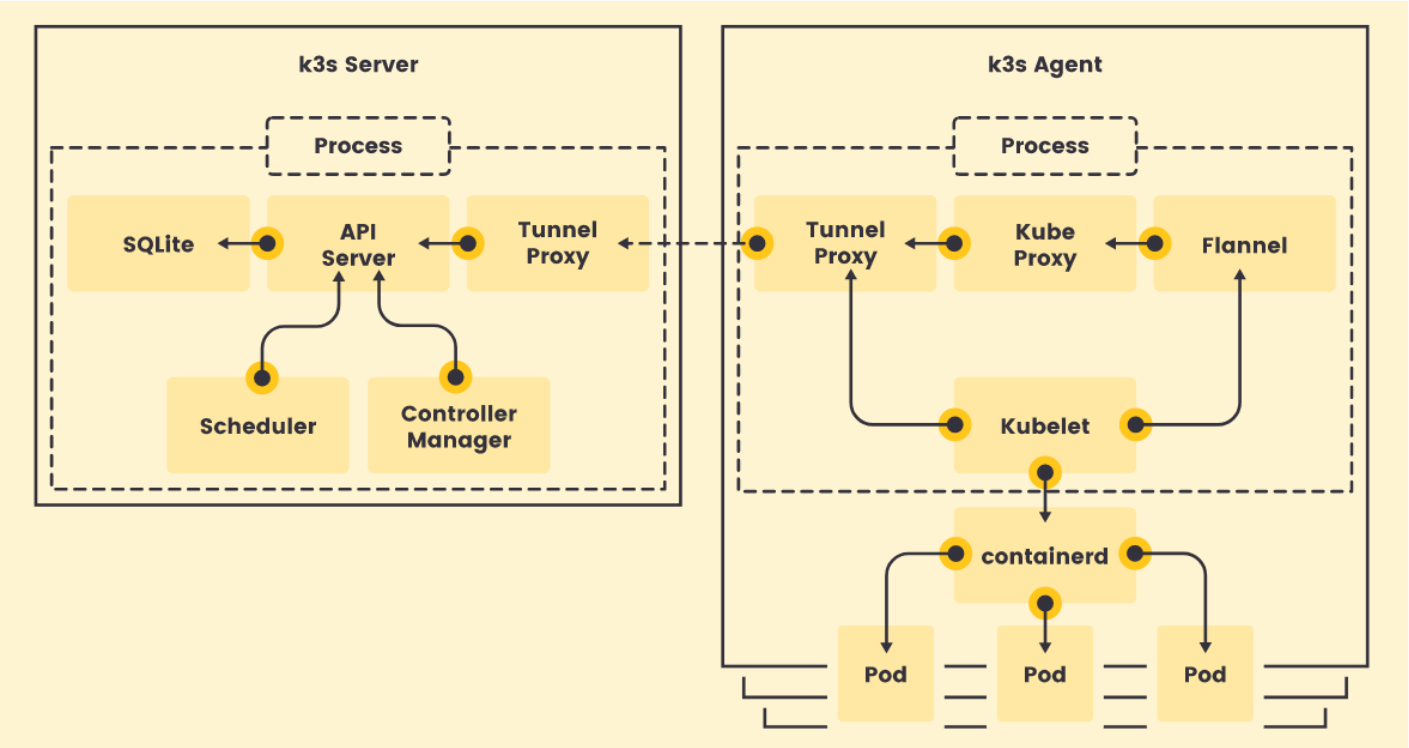
What's included in K3s

- Containerd
- Flannel
- CoreDNS
- CNI
- Host utilities (iptables, socat, etc)
- Ingress controller (traefik)
- Embedded service loadbalancer
- Embedded network policy controller

How to use K3s

- Uses per default Containerd as container-engine
- Can use alternatively Docker, but it's not required
- Run's as a Server and a Node on the same machine
- But also as Server(s) and Node(s) on separate machines
- You need at least one Server and one Node
- For high availability K3s supports a cluster of multiple servers

The architecture of K3s



Install K3s is very easy

Install the server

```
3  K3S_DATA_DIR=/data/k3s
4
5  export K3S_KUBECONFIG_MODE=644
6  export K3S_TOKEN="qa-k3s-cluster-1"
7
8  export INSTALL_K3S_EXEC="server --docker --data-dir ${K3S_DATA_DIR}"
9
10 curl -sfL https://get.k3s.io | sh -
```

Install the agent

```
export K3S_TOKEN="qa-k3s-cluster-1"

K3S_URL="https://[redacted]:6443"
K3S_DATA_DIR=/data/k3s

export INSTALL_K3S_EXEC="agent --server ${K3S_URL} --data-dir ${K3S_DATA_DIR} --docker"

curl -sfL https://get.k3s.io | sh
```

K3s will be installed as Systemd service

Server

```
● k3s.service - Lightweight Kubernetes
   Loaded: loaded (/etc/systemd/system/k3s.service; enabled; vendor preset: enabled)
   Active: active (running) since Sat 2020-07-11 12:32:12 CEST; 2 weeks 2 days ago
     Docs: https://k3s.io
 Main PID: 1232 (k3s-server)
    Tasks: 0
   CGroup: /system.slice/k3s.service
           └─1232 /usr/local/bin/k3s server --docker --node-label agent-type=server --data-dir /data/k3s
```

Agent

```
● k3s-agent.service - Lightweight Kubernetes
   Loaded: loaded (/etc/systemd/system/k3s-agent.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2020-07-24 15:50:56 CEST; 3 days ago
     Docs: https://k3s.io
 Main PID: 1328 (k3s-agent)
    Tasks: 0
   CGroup: /system.slice/k3s-agent.service
           └─1328 /usr/local/bin/k3s agent --docker --node-label agent-type=worker --data-dir /data/k3s
```

The whole K3s cluster

State	Name	Roles	Version	CPU	RAM	Pods
Active	bu-int-k8s-node-01 172.30.136.197 agent-type=worker	Worker	v1.18.3+k3s1 19.3.11	0.9/4 Cores	1.7/7.8 GiB	36/110
Active	bu-int-k8s-node-02 172.30.136.198 agent-type=worker	Worker	v1.18.3+k3s1 19.3.11	0.9/4 Cores	1.6/7.8 GiB	34/110
Active	bu-int-k8s-node-03 172.30.136.199 agent-type=worker	Worker	v1.18.3+k3s1 19.3.11	0.9/4 Cores	1.1/7.8 GiB	24/110
Active	bu-int-k8s-node-04 172.30.136.200 agent-type=worker	Worker	v1.18.3+k3s1 19.3.11	0.8/4 Cores	1.7/7.8 GiB	37/110
Active	bu-int-k8s-node-05 172.30.136.205 agent-type=worker	Worker	v1.18.3+k3s1 19.3.11	0.6/4 Cores	1/7.8 GiB	23/110
Active	bu-int-k8s-server-01 172.30.136.196 agent-type=server	Control Plane	v1.18.3+k3s1 19.3.11	0.1/2 Cores	0.1/3.9 GiB	7/110

What's is the role of Rancher

- Makes the access to the cluster easier. (UserManagement)
- Can configure monitoring with Prometheus and Grafana
- Works fine together with K3s because it's from the same company
- Easy version upgrades for the K3s cluster with the system-upgrade-controller
- Easier access to container logs and analyzing deployment problems

Questions?

Thank you!