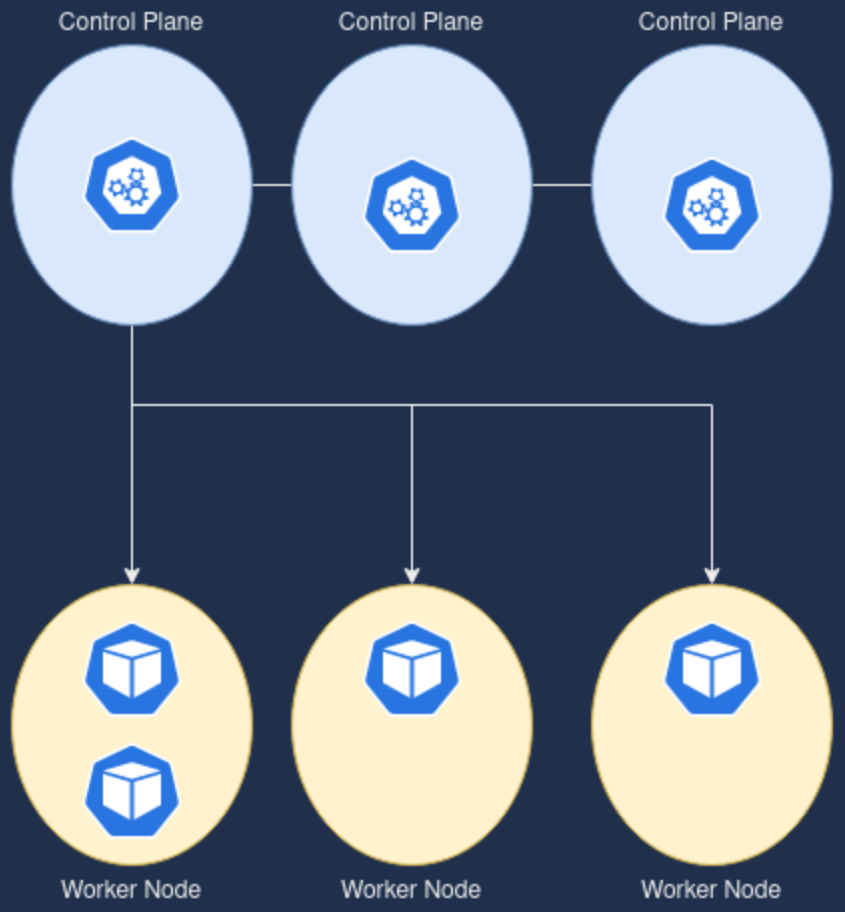
Kubernetes: Betrieb verschiedener Workloads

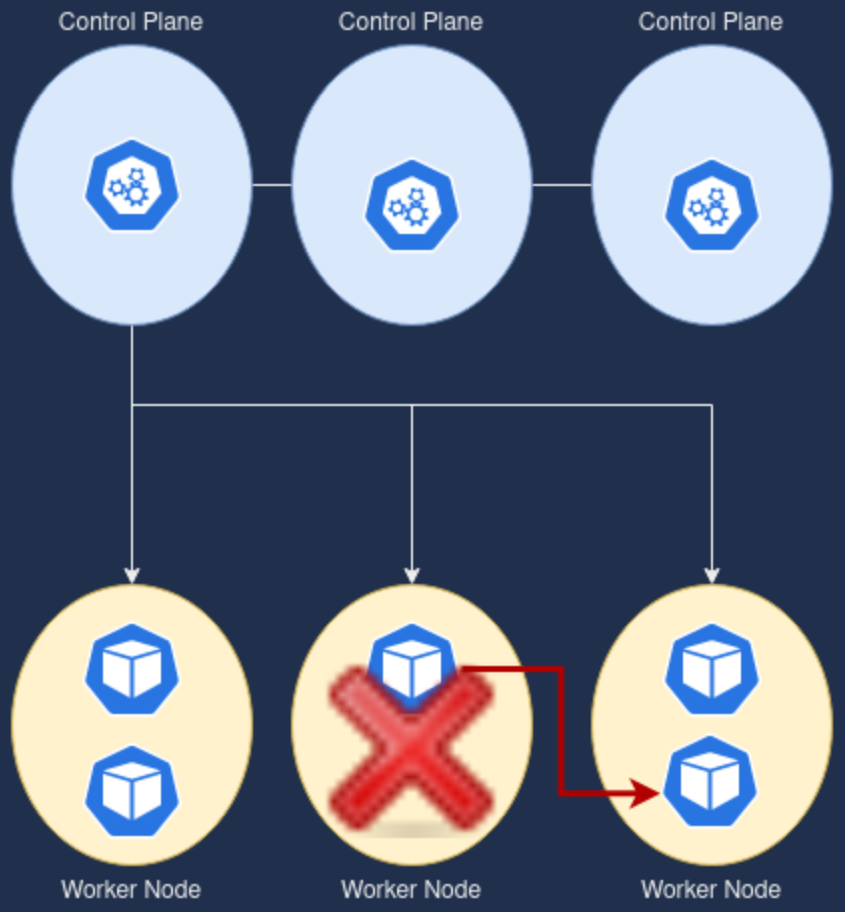
Kurze Einführung: Was ist Kubernetes?

- Kubernetes ist eine Orchestrierungslösung für Container
- Container werden in "Pods" gruppiert
  - Ein Pod beinhaltet einen oder mehrere Container die immer zusammen gestartet werden
  - z.B. Anwendungscontainer und Logging Sidecar
  - bei einfachen Apps oft einfach ein Container
- Pods werden gestartet / gestoppt
  - Mit einem *Deployment* legt man fest wie ein Pod aussieht und wie oft dieser gestartet wird
- Pods werden verteilt und in der gewünschten Anzahl gestartet
- Pods werden untereinander vernetzt (auch über Nodes Server/Nodes hinweg)
- Pods werden bei Bedarf von außen erreichbar gemacht

Verteilte Systeme mit Kubernetes



Verteilte Systeme mit Kubernetes



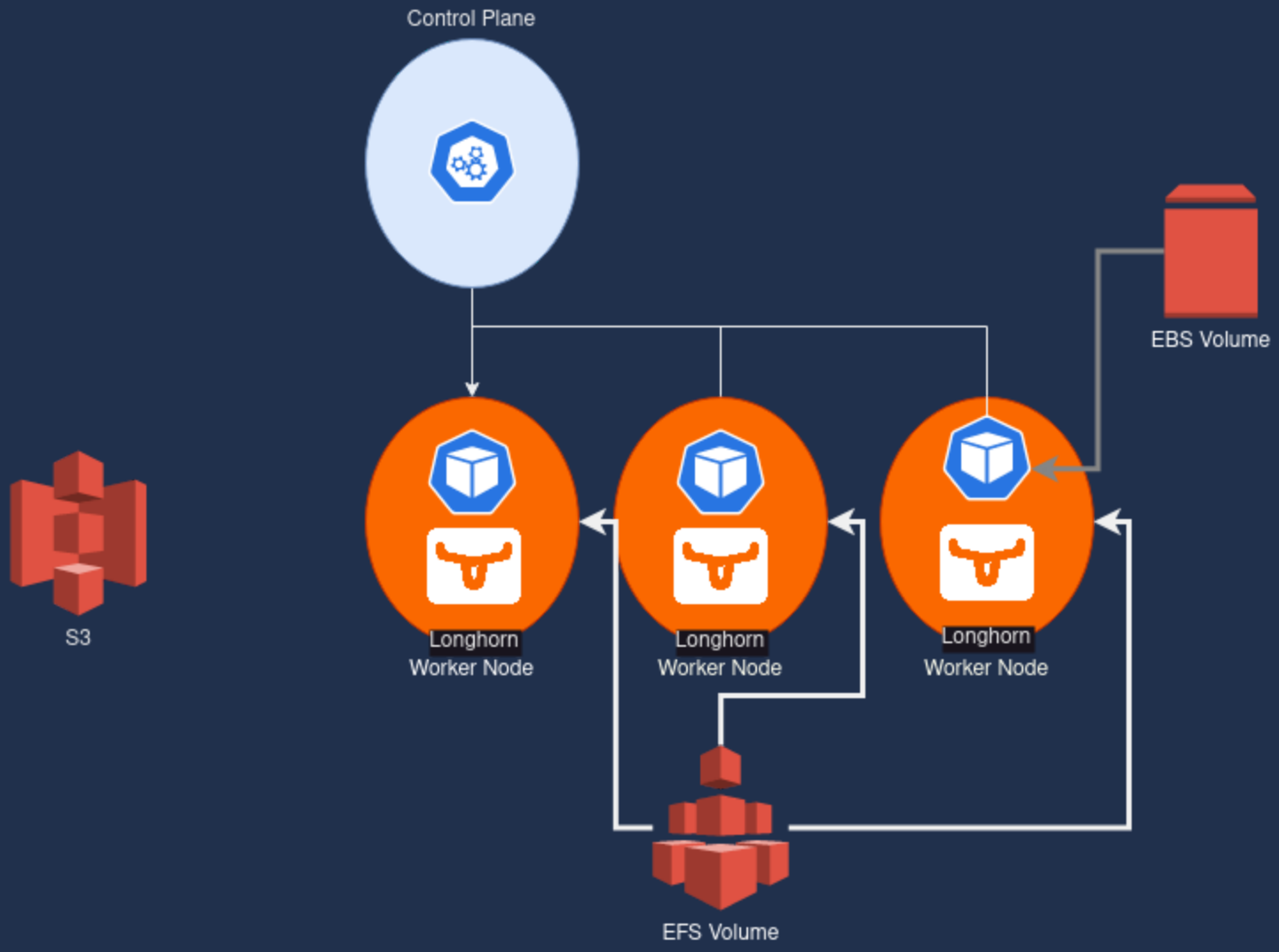
Welche Workloads eignen sich für das gezeigte Schema?

- Stateless Workloads
- Kein speichern von Daten auf den Nodes
- Weder für Client noch Server darf die Instanz die läuft entscheidend sein
- Der Container/Pod oder der ganze Server können jederzeit aus dem Cluster verschwinden

Meine Anwendung muss aber Daten persistieren

- Externer Objectstore (z.B. *AWS S3*) für Assets
- Externes Netzwerkdateisystem (*NFS*, *AWS EFS*, *GlusterFS*, ...)
- Verteiltes Dateisystem im Kubernetes (z.B. *Longhorn*)
- Externe Datenbank (auf eigenen VMs, *AWS RDS*)
- Datenbank im Cluster replizieren
  - Vorsicht bei großen Datenbanken
  - je nach Datenbank Downtimes beim Failover
  - Block Storage für Datenbanken dynamisch an Nodes anhängen um unnötige Replikation zu vermeiden
    - Muss aber vom Storage Provider unterstützt werden (z.B. *AWS EBS*)

Persistenz Lösungen erklärt

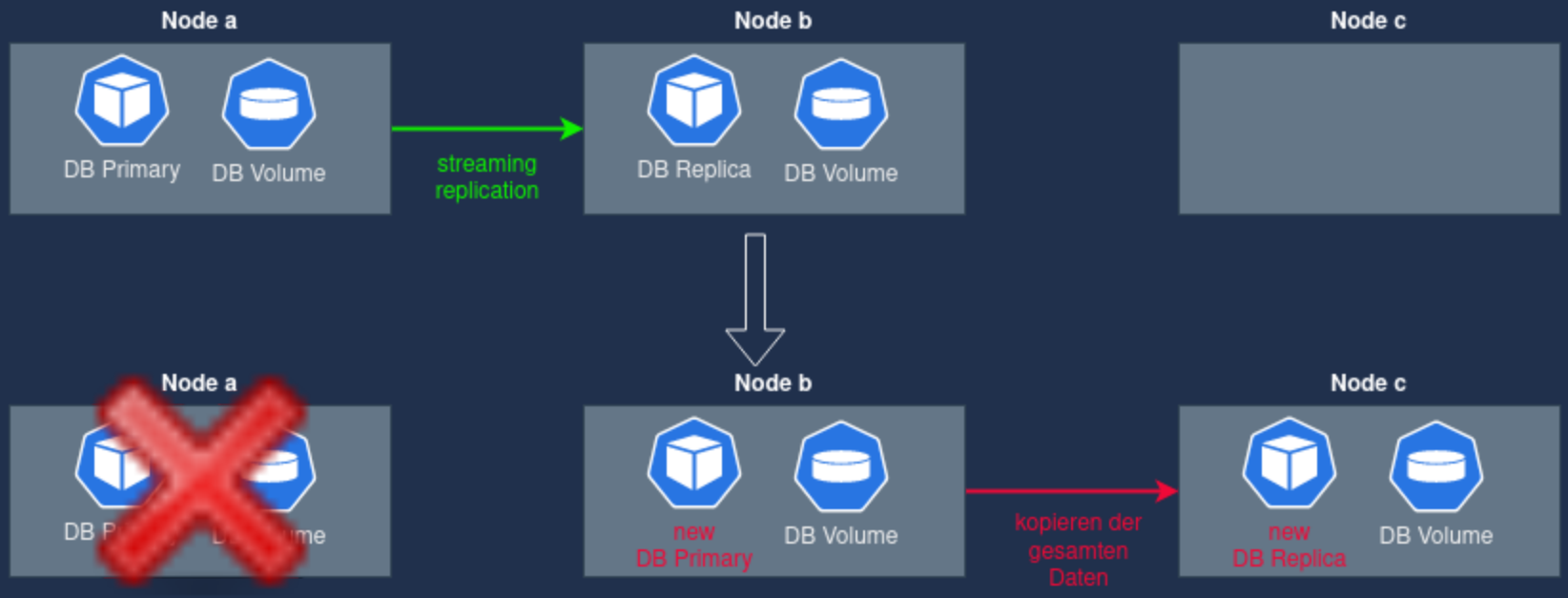


Also was eignet sich für was?

- Webassets (Bilder, Fonts, ..): *Object Storage*, *Netzwerk Dateisystem* (*NFS* oder *Longhorn*)
  - Assets müssen von mehreren Pods lesbar / schreibbar sein
- Cache für regelmäßige Jobs oder CI Jobs: *Object Storage*, *Netzwerk Dateisystem*
  - Cache muss von mehreren Pods lesbar / schreibbar sein
- Datenbank Storage: *Dynamic Block Storage*, *Local Block Storage* (abhängig von Setup des Systems)
  - Hier wird höhere Performance erwartet. Außerdem ist es nicht nötig, dass mehrere Pods hier lesen und schreiben können
  - Statt eines *Deployments* verwendet man ein *StatefulSet*
    - Kubernetes weiß, dass die erstellten Pods State haben, der z.B. in einem Volume liegt das dem Pod fest zugeordnet wird (z.B. Pod *psql-1* hat ein volume *psql-1* das immer diesem Pod zugeordnet wird)

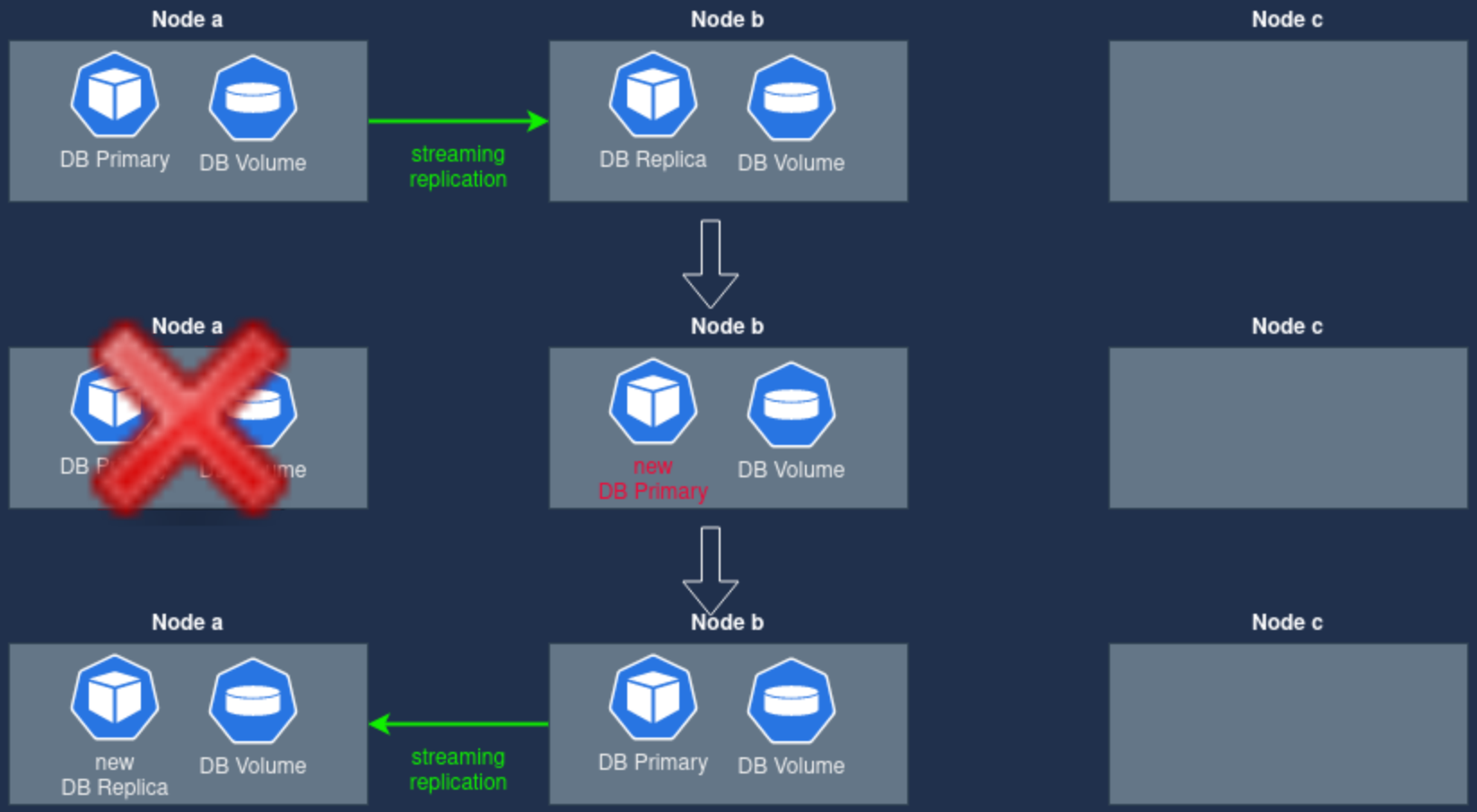
Datenbank mit local volumes und Deployment

Keine empfehlenswerte Lösung.

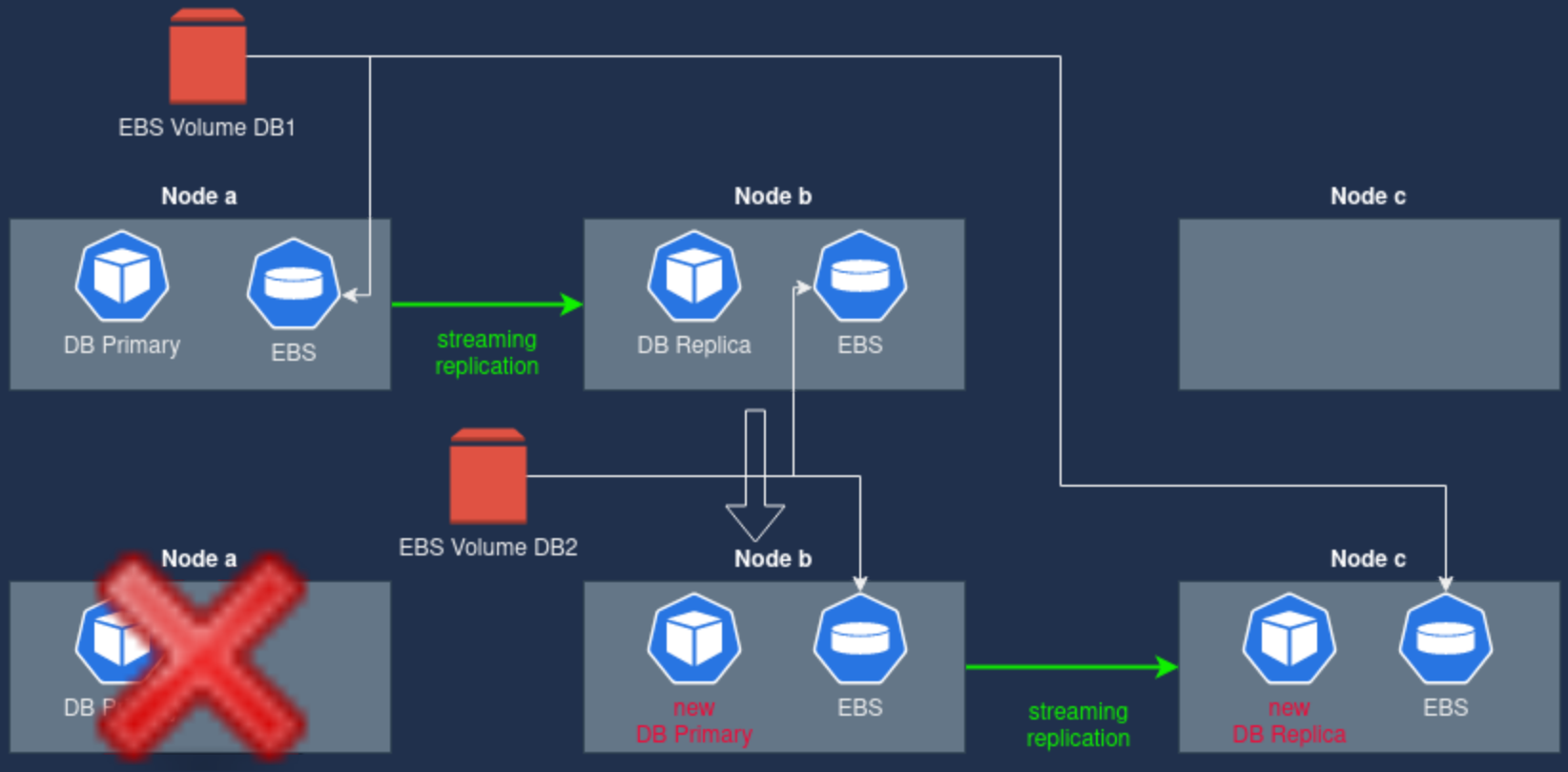


Datenbank mit local volumes und StatefulSet

Besser, solange man kein Autoscaling verwendet. **Aber**: Kein Redundanz solange eine Node fehlt.



Datenbank mit EBS volumes und StatefulSet



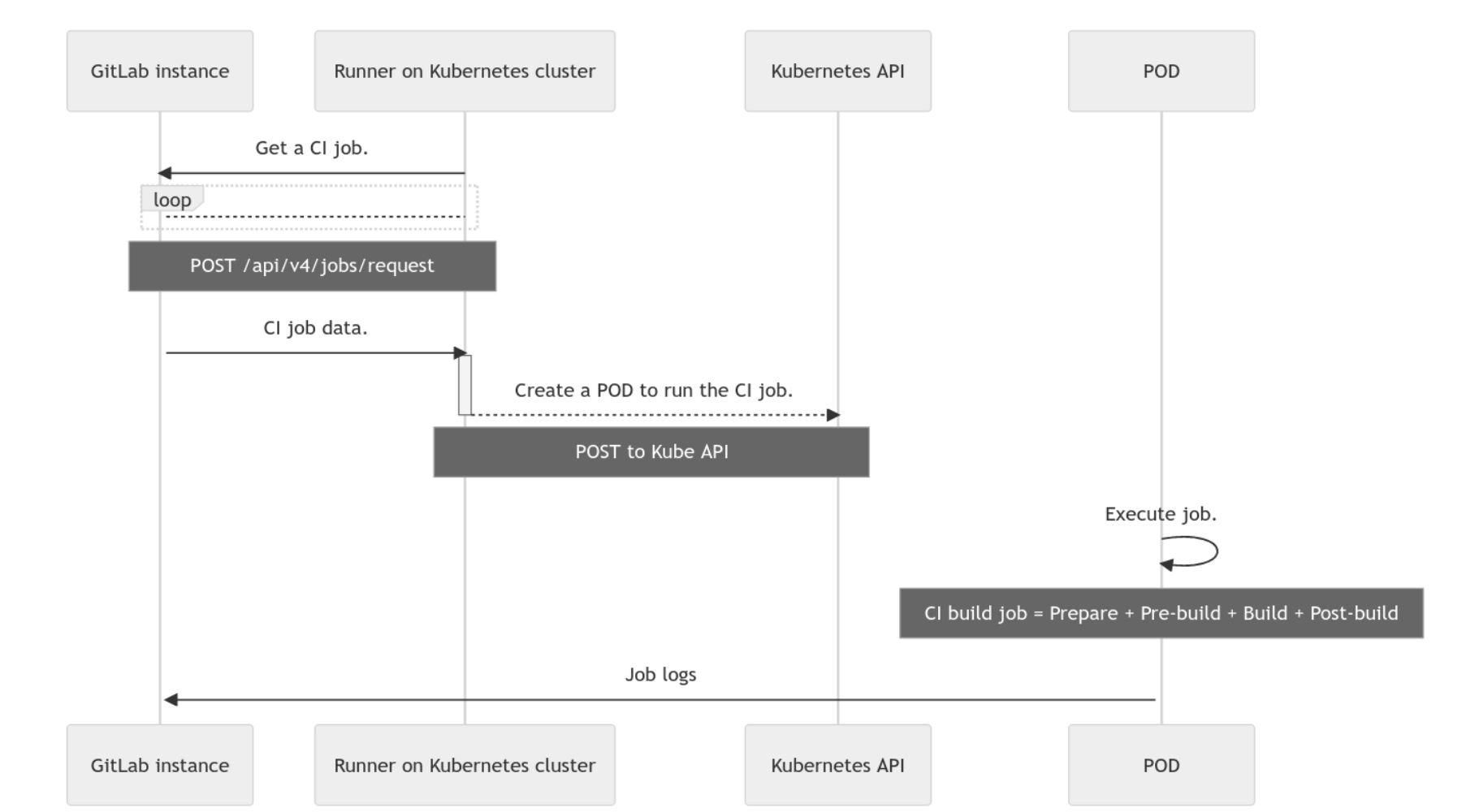
Autoscaling und unterschiedliche Compute Anforderungen

- Unterschiedliche Workloads haben unterschiedliche Anforderungen
- Beispiele:
  - Testumgebungen mit wenig Last möchte man möglichst kostensparend betreiben
  - Relationale Datenbanken sollten Failover möglichst vermeiden
  - Pods die (langlaufende) Jobs ausführen sollten dabei möglichst nicht unterbrochen werden
  - Manche Jobs sind besonders CPU intensiv und brauchen entsprechende CPU Ressourcen

AWS EC2 Ressourcen nach Bedarf optimieren

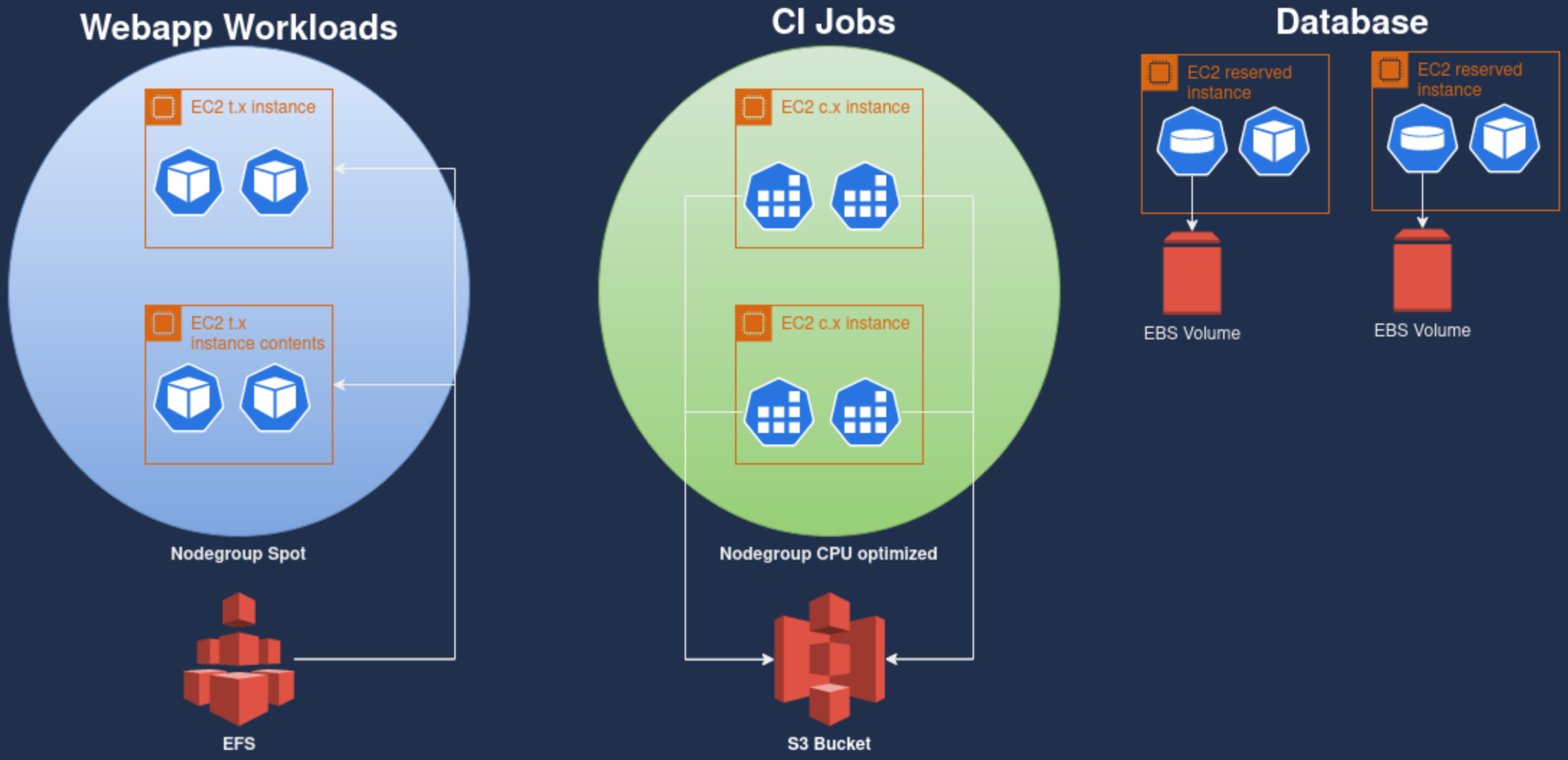
- Autoscaling Groups um Compute Resources nach Bedarf zu skalieren
  - Instanztypen festlegen
  - Scaling Optionen festlegen (oder Kubernetes *Cluster Autoscaler* verwenden)
- Geld sparen bei Stateless Workloads mit *Spot Instances*
  - AWS bietet Compute Ressourcen die gerade nicht aktiv genutzt werden günstig an
  - Bei *OnDemand* Bedarf werden die Spot Instances automatisch beendet
- EC2 Instanzen außerhalb von Autoscaling für dauerhaft laufende Stateful Anwendungen (z.B. Datenbank)

Kurzer Exkurs zu "Jobs" mit Gitlab CI



Quelle: https://docs.gitlab.com/runner/executors/kubernetes.html

Workload Anforderungen erfüllen



Umsetzung der Storage Bereitstellung

- Zugriff auf S3 erfolgt über die S3 API. Credentials können über OIDC an die jeweiligen Applikations Pods gegeben werden (darauf gehen wir nicht weiter ein)
- EBS und EFS Volumes werden über Kubernetes *Storage Classes* und *Storage Driver* eingebunden / bereitgestellt.
  - Der *Storage Driver* bzw. *Provisioner* (für z.B. EBS) sort dafür, dass beim Anfragen einer bestimmten *Storage Class* die diesen verwendet das Volume erstellt wird.
  - Volumes werden als *Physical Volume* im Kubernetes repräsentiert und mit einem *Physical Volume Claim* angefragt.

EBS Storageclass

Der Provider für EBS Volumes und die API Permissions für diesen müssen schon vorhanden sein.

```
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
...
```

EBS PVC

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: database-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: ebs-sc
  resources:
    requests:
      storage: 50Gi
```

EBS Pod

```
---
apiVersion: v1
kind: Pod
metadata:
  name: database-a
spec:
  containers:
  - name: database
    image: registry.foo/database-image:v1.0.0
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: database-claim
```

EFS StorageClass

Der Provider für EFS Volumes und ein EFS Volume müssen bereits vorhanden sein.

```
---
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: efs-sc
provisioner: efs.csi.aws.com
...
```

EFS PVC

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: app-storage
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: efs-sc
  resources:
    requests:
      storage: 50Gi
```

EFS Pod

```
---
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
    - name: app
      image: registry.foobar/app-image:v1.0.0
      volumeMounts:
        - name: persistent-storage
          mountPath: /assets
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: app-storage
```

Zuweisen der Pods zu den richtigen Nodes

- statische EC2 Instanzen für die Datenbank
- eine *Nodegroup* für die Anwendungscontainer
- eine *Nodegroup* für unsere Jobs
- **Die Frage:** Wie landen die Pods/Container auf den richtigen Nodes?

Labels, Selectors, Taints und Tolerations

Kubernetes Nodes haben **label**:


```
{
  "beta.kubernetes.io/arch": "amd64",
  "beta.kubernetes.io/instance-type": "c6i.xlarge",
  "beta.kubernetes.io/os": "linux",
  "eks.amazonaws.com/capacityType": "ON_DEMAND",
  "eks.amazonaws.com/nodegroup": "test-cluster-compute_demand-20230314160827706400000001",
  "eks.amazonaws.com/nodegroup-image": "ami-04dc8cdc2e948f054",
  "eks.amazonaws.com/sourceLaunchTemplateId": "lt-08498f74049e4e277",
  "eks.amazonaws.com/sourceLaunchTemplateVersion": "1",
  "failure-domain.beta.kubernetes.io/region": "eu-west-1",
  "failure-domain.beta.kubernetes.io/zone": "eu-west-1b",
  "k8s.io/cloud-provider-aws": "9650fdbb8a8b5bfe4fd8cf1ea1de26c9",
  "kubernetes.io/arch": "amd64",
  "kubernetes.io/hostname": "ip-10-0-32-26.eu-west-1.compute.internal",
  "kubernetes.io/os": "linux",
  "node.kubernetes.io/exclude-from-external-load-balancers": "true",
  "node.kubernetes.io/instance-type": "c6i.xlarge",
  "nodegroup_name": "compute_demand",
  "topology.kubernetes.io/region": "eu-west-1",
  "topology.kubernetes.io/zone": "eu-west-1b"
}
```

Labels, Selectors, Taints und Tolerations

Node **labels** können verwendet werden um zu bestimmen auf welchen Nodes Pods starten.

- `Selectors`: Pods starten nur auf einer Node auf dem die Labels matchen
- `Taints` und `Tolerations`: Hat eine node `Taints` starten hier nur Pods die eine entsprechende `Toleration` dafür haben.

Selectors

Wenn wir unsere Jobs nur auf den Compute Instanzen laufen lassen möchten, konfigurieren wir diese mit einem entsprechenden Selector.

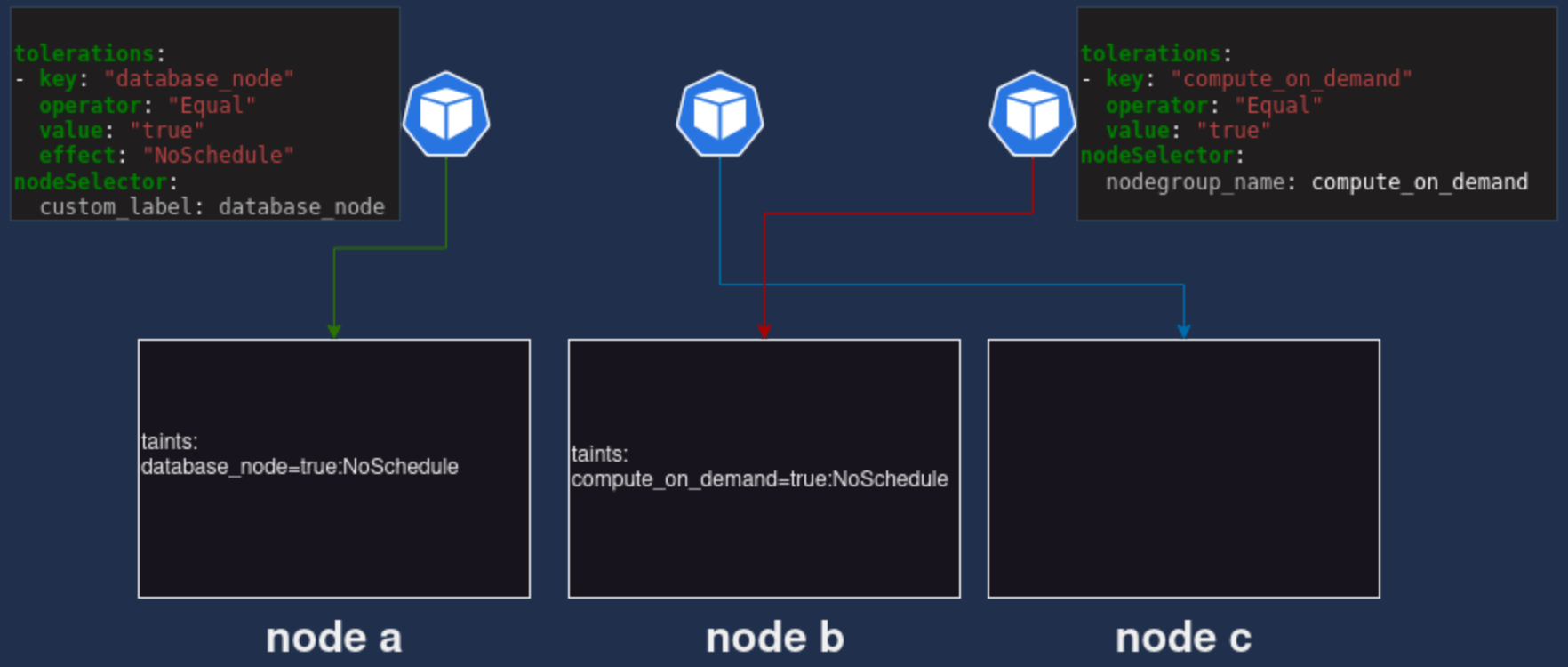
Weitere Selectors

- Für die Datenbank Instanzen ist es wichtig, dass diese auf Instanzen außerhalb der ASG starten
- Selector für die Datenbank Pods hinzufügen, der auf die statischen Instanzen zeigt.

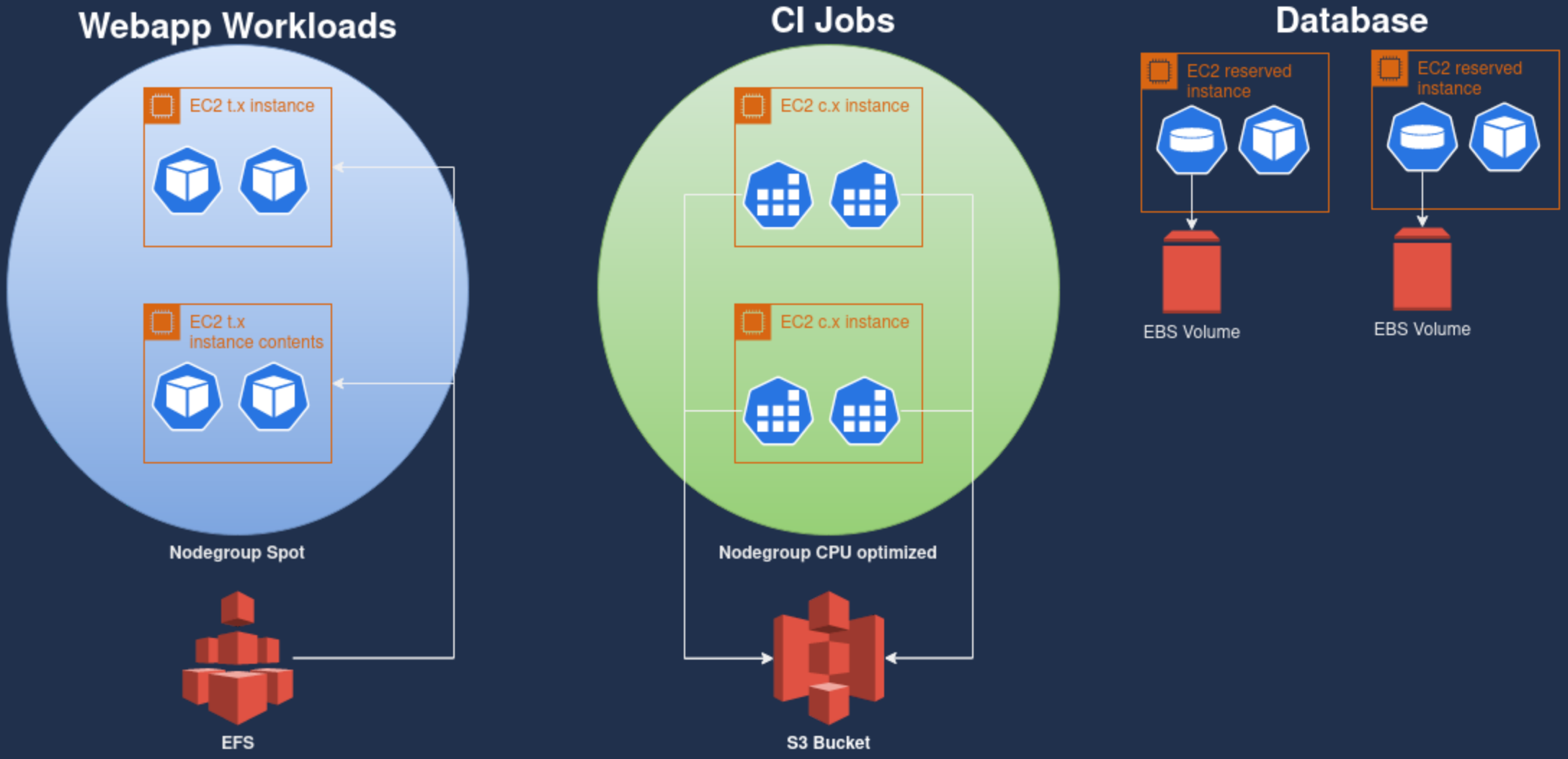
Taints und Tolerations

- Bisheriges Setup sorgt dafür, dass Jobs und Datenbanken auf den richtigen Nodes starten
- Applikations Pods starten im Moment noch auf beliebigen Nodes (Spot, Compute, DB)
- Wir könnten einen Selector hinzufügen, aber was wenn jemand bei neuen Workloads vergisst?

Taints und Tolerations



Ziel erreicht!



Vielen Dank für die Aufmerksamkeit

Zu kompliziert?

makandra hilft euch gerne weiter.

Wissensdurstig?

Werde Trainee bei makandra.