**1. Grundlagen von Kubernetes (K8s) und Microservices**

* **Kubernetes (K8s)** ist eine Open-Source-Plattform zur Automatisierung der Bereitstellung, Skalierung und Verwaltung containerisierter Anwendungen. Sie bietet eine verwaltete Umgebung, um Anwendungen in Containern zu betreiben und die Infrastruktur effizient zu nutzen.
* **Microservice-Architektur** ist ein Designansatz, bei dem Anwendungen als Sammlung unabhängiger Dienste organisiert sind. Jeder Microservice ist für eine bestimmte Funktion verantwortlich und kann unabhängig entwickelt, bereitgestellt und skaliert werden.
* **K8s-Komponenten**:
  + **Master-Komponenten** (API-Server, etcd, Controller Manager, Scheduler) verwalten den Cluster und koordinieren die Nodes.
  + **Node-Komponenten** (Kubelet, Kube-Proxy, Container Runtime) betreiben die Container und kommunizieren mit dem Master.
* **Übungen**: Arbeiten mit grundlegenden Kubernetes-Befehlen, wie dem Erstellen von Pods, Überprüfen von Logs, und Analysieren von Fehlermeldungen.

**2. Zugriff auf EC2-Instanzen über SSH**

* **Ziel**: Verbindungsaufbau zu EC2-Instanzen mit SSH.
* **SSH-Verbindung** ist ein sicherer Protokollzugriff auf Server, der eine verschlüsselte Kommunikation ermöglicht. Durch den SSH-Zugriff auf eine EC2-Instanz können Kubernetes-Befehle direkt in der Instanz ausgeführt werden.
* **Prozess**: SSH-Schlüssel generieren, den öffentlichen Schlüssel auf die EC2-Instanz laden und die Verbindung über den privaten Schlüssel herstellen.

**3. Monolithische vs. Microservice-Architekturen**

* **Monolithische Architektur**: Eine klassische Architektur, bei der alle Funktionen einer Anwendung in einer einzigen, großen Codebasis zusammengefasst sind. Änderungen sind aufwendiger, da alle Teile miteinander verbunden sind.
  + **Vorteile**: Einfacheres Debugging und Deployment, klare Struktur bei kleinen Projekten.
  + **Nachteile**: Schwieriger zu skalieren, unflexibel, komplexe Fehlerbehebung bei großen Projekten.
* **Microservice-Architektur**: Eine moderne, verteilte Architektur, die es erlaubt, Anwendungen in kleinere, unabhängige Dienste zu unterteilen, die einzeln verwaltet werden können.
  + **Vorteile**: Flexibles Skalieren, individuelle Entwicklung und Bereitstellung von Services, einfachere Wartung.
  + **Nachteile**: Komplexeres Management der Kommunikation zwischen Services, erhöhter Netzwerkaufwand, potenziell höhere Kosten.
* **Anwendungsfälle**: Microservices eignen sich gut für große, dynamische Anwendungen, die häufig skaliert und aktualisiert werden müssen (z. B. E-Commerce-Websites).

**4. App-Bereitstellung in einem K8s-Cluster**

* **Deployment** einer App in Kubernetes bedeutet das Erstellen und Managen einer Reihe von Pods und ReplicaSets, die bestimmte Versionen einer App enthalten.
* **Minikube** ist ein Tool, das das lokale Ausführen von Kubernetes-Clustern ermöglicht, ideal zum Testen und Entwickeln.
* **Dashboard**: Kubernetes bietet ein Dashboard, das eine visuelle Übersicht des Clusters bietet, mit dem man Pods, Nodes und andere Ressourcen überwachen kann.
* **Kommandos**: Mit grundlegenden Befehlen wie kubectl create, kubectl get, kubectl describe und kubectl delete lassen sich Deployments steuern und Informationen abrufen.
* **Übungen**: Deployment einer Anwendung im Cluster, Verständnis der K8s-Architektur und Fehleranalyse bei Fehlermeldungen.

**5. App-Expose in Kubernetes**

* **Service**: Ein Kubernetes-Service stellt eine dauerhafte IP-Adresse für Pods bereit und erlaubt es, eine Verbindung zu ihnen herzustellen. Verschiedene Service-Typen (ClusterIP, NodePort, LoadBalancer) bestimmen, wie der Zugriff von außen erfolgt.
* **Labels und Selectors**: Mit Labels werden Objekte im Cluster gekennzeichnet, um sie einfacher verwalten und organisieren zu können. Selectors helfen dabei, bestimmte Gruppen von Objekten zu identifizieren und zu steuern.
* **Expose** einer Anwendung bedeutet, sie für den externen Zugriff verfügbar zu machen, meist durch das Erstellen eines LoadBalancer- oder NodePort-Services.
* **Übungen**: Deployment und Exponierung einer App im K8s-Cluster, Auswahl und Verwaltung von Pods durch Labels.

**6. App-Skalierung in Kubernetes**

* **Skalierung** bedeutet, die Anzahl von Instanzen einer App zu erhöhen oder zu verringern, um den Ressourcenverbrauch zu optimieren.
  + **Horizontale Skalierung**: Fügt zusätzliche Pods hinzu, um die Verfügbarkeit zu erhöhen.
  + **Vertikale Skalierung**: Erhöht die Ressourcen eines einzelnen Pods (z. B. Speicher und CPU).
* **ReplicaSet**: Kubernetes steuert die Anzahl der gewünschten Pod-Instanzen durch das ReplicaSet. Wenn ein Pod ausfällt, erstellt das ReplicaSet automatisch einen neuen Pod, um die gewünschte Anzahl beizubehalten.
* **Übungen**: Skalieren einer Anwendung in einem Kubernetes-Cluster, Beobachten und Verifizieren des Skalierungsprozesses, Analysieren und Lösen von Problemen, die durch unzureichende Ressourcen entstehen können.

Wichtigste commands

Create a cluster with minikube

minikube start #??

Deploy the app with the name bzz-app-3

kubectl create deployment bzz-app-3 --image=gcr.io/google-samples/kubernetes-bootcamp:v1

Display the deployment

kubectl get deployments #??

1. Create a cluster with minikube

minikube start #??

1. Deploy the mentioned app. Give it the name bzz-k8s-app-1

kubectl create deployment bzz-k8s-app-1 --image=gcr.io/google-samples/kubernetes-bootcamp:v1 #??

1. Display the deployment

kubectl get deployments #??

1. View the app with kubectl proxy. For that you need to open a second terminal window to run the proxy. Try now i.e. to query the version directly through the API with curl. Background: The kubectl proxy command can create a proxy that will forward communications into the cluster-wide, private network. The proxy can be terminated by pressing control-C and won't show any output while it's running.

kubectl proxy --port=8001 #??

curl http://localhost:8001/version #??

1. Access the Pod through the proxied API. Get first the Pod name, and store it in the environment variable POD\_NAME

export POD\_NAME=$(\

kubectl get pods #??\

-o go-template --template \

'{{range .items}}{{.metadata.name}}{{"\n"}}{{end}}')

Background: The API server will automatically create an endpoint for each pod, based on the pod name, that is also accessible through the proxy.

1. Check the content of the POD\_NAME

echo ${POD\_NAME} #??

1. Access the Pod through the proxied API (<http://localhost:8001/api/v1/namespaces/default/pods/><POD-NAME>:8080/proxy/) with curl.

curl http://localhost:8001/api/v1/namespaces/default/pods/$POD\_NAME:8080/proxy/ #??