Tools	Minikube, Dockerclient, Kops, Kubeadm
Minikube	minikube delete minikube start minikube version minikube dashboard minikube status minikube ip minikube addons list minikube service {service} --url
Kubectl	kubectl api-versions kubectl cluster-info https://kubernetes.io/docs/reference/kubectl/cheatsheet/
Deployment	# replication controller #replication set # deployment https://www.bmc.com/blogs/kubernetes-deployment/ kubectl create -f deployment/helloworld.yml kubectl get deployments or kubectl get deploy kubectl get service,pvc,deployment,pods kubectl get all | grep hello-world kubectl get rs or kubectl get replicaset kubectl rollout status deployment/helloworld-deployment kubectl rollout history deployment kubectl set image deployment/helloworld-deployment k8s-demo=wardviaene/k8s-demo:2 kubectl rollout status deployment/helloworld-deployment curl http://192.168.99.103:30353 kubectl rollout history deployment kubectl scale --replicas=4 deployment helloworld-deployment kubectl rollout undo deployment/helloworld-deployment kubectl rollout undo deployment/helloworld-deployment —to-revision=3 **kubectl delete deployment hello-world** ReplicaSet is the next-generation Replication Controller. we recommend using Deployments instead of directly using ReplicaSets, unless you require custom update orchestration or don’t require updates at all.This actually means that you may never need to manipulate ReplicaSet objects: use a Deployment instead, and define your application in the spec section.
Pod	kubectl get pods —show-labels kubectl describe pod myPodName -n myNameSpace

Service	kubectl expose deployment helloworld-deployment --type=NodePort --name=example-service
	
	This is equivalent to **kubectl create -f** the following yaml:
	apiVersion: v1
	kind: Service
	metadata:
	  name: example-service
	spec:
	  ports:
	  - port: 31001
	  nodePort: 31001
	  targetPort: nodejs-port
	  protocol: TCP
	  selector:
	  app: helloworld
	  type: NodePort
	
	Kubectl get service
	kubectl describe service example-service
	minikube service example-service --url
	curl http://192.168.99.103:30353
	
	A Kubernetes Service is an abstraction which defines a logical set of Pods running somewhere in your cluster, that all provide the same functionality. When created, each Service is assigned a unique IP address (also called clusterIP). This address is tied to the lifespan of the Service, and will not change while the Service is alive. Pods can be configured to talk to the Service, and know that communication to the Service will be automatically load-balanced out to some pod that is a member of the Service.
	
	ClusterIP: makes the service only reachable from within the cluster. This is the default ServiceType.
	
	NodePort : Kubernetes will allocate a specific port on each Node to that service, and any request to your cluster on that port gets forwarded to the service.
	Limitation: have to take care of load balancing so that you leverage the power of your pod replicas. if you want to manage load balancing systems manually and set up port mappings yourself.
	
	LoadBalancer : There needs to be some external load balancer functionality in the cluster, typically implemented by a cloud provider.
	This is typically heavily dependent on the cloud provider.Every

	Kubectl delete svc example-service
LivenessProbe RedinessProbe	liveness probes to know when to restart a Container. readiness probes to know when a Container is ready to start accepting traffic.
Config	http://pwittrock.github.io/docs/user-guide/configmap/ https://medium.com/google-cloud/kubernetes-configmaps-and-secrets-68d061f7ab5b **kubectl create -f configmap.yaml** kubectl create configmap language --from-literal=LANGUAGE=English **kubectl get configmap** **kubectl describe configMap test-configmap** **kubectl get configmaps test-configmap -o yaml**
Secret	kubectl create secret generic apikey --from-literal=API_KEY=123–456 kubectl get secret

	Before developers can start using the storage, administrators need to provision persistent volumes. Unlike volumes, persistent volumes are not associated with any specific pod or containers when they are created. They are pre-provisioned storage resources that can be used by developers during the creation of a pod. Once persistence volumes (PersistentVolume) are provisioned by administrators, developers create a claim (PersistentVolumeClaim) to start consuming the storage resources exposed as persistent volumes. **Volume** When running multiple containers together in a Pod it is often necessary to share files between those containers; volumes are used for this purpose. PERSISTENT VOLUME A PersistentVolume (PV) is a piece of storage in the cluster. It is a resource just like a node is a cluster resource. PVs are volume plugins like volumes but have a lifecycle independent of any individual pod that uses the PV. Volumes will be deleted when Pods are deleted, while Persistent Volumes are different entities, completely decoupled from the Pod. PVs are managed by a different set of APIs \ kubectl than Pods and have their own Lifecycle. PERSISTENT VOLUME CLAIMS A PersistentVolumeClaim (PVC) is a request for storage. It is similar to a pod. While Pods consume node resources, PVCs consume Persistent Volume resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., can be mounted once read/write or read-only).

Volume	**minikube ssh** Sql https://kubernetes.io/docs/tasks/run-application/run-single-instance-stateful-application/ https://dev.mysql.com/doc/mysql-getting-started/en/ if you get this error Can't create database 'database'. (errno: 13)' http://www.webhostingtalk.com/showthread.php?t=361700 change to the directory above the mysql dir, probably var/lib/: **# cd /var/lib** Check the owner and group with **# ls -ld mysql** **# chown -R mysql:mysql mysql** Nginx https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/ Wordpress https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/ https://dotlayer.com/how-to-run-wordpress-in-a-kubernetes-cluster/

Ingress	it sits in front of multiple services and act as a “smart router” or entrypoint into your cluster. Ingress Controllers can technically be any system capable of reverse proxying, but the most common is Nginx. It is a completely independent resource to your service. You declare, create and destroy it separately to your services. https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html https://velotio.com/blog/2017/7/5/http-load-balancing-in-kubernetes-with-ingress https://medium.com/@awkwardferny/getting-started-with-kubernetes-ingress-nginx-on-minikube-d75e58f52b6c minikube addons enable ingress # wait a min for the pod to be up and running $ kubectl get pods -n kube-system | grep nginx-ingress-controller Kubernetes ingress resource lacks of quite important functionality, mainly mutual TLS for incoming connections (currently only support server certificates), supporting SNI (releated with the previous one), non-HTTP traffic load balance, and if you want to use Istio features like Traffic splitting, fault injection, mirroring, header match, etc are not available between ingress controller and backend... I was looking for for an implementation like this, let see if "gateway" finally works and performance is as good as ingress controller.

Service discovery	Kubernetes DNS schedules a DNS Pod and Service on the cluster, and configures the kubelets to tell individual containers to use the DNS Service's IP to resolve DNS names. **kubectl get services kube-dns --namespace=kube-system** https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough You can expose your application via a **ClusterIP** service.**ClusterIP** is the default **ServiceType** and it creates a single IP address that can be used to access its **Pods** which can only be accessed from inside the cluster. If **KubeDNS** is enabled it will also get a series of DNS records assigned to it include an A record to match its IP. This is very useful for exposing microservices running inside the same Kubernetes cluster to each other. kubectl run -i --tty busybox --image=busybox --restart=Never -- sh #nslookup azure-vote-back Kubectl get pods kubectl exec -it azure-vote-front-fc749d98c-h642z -- /bin/bash #ping azure-vote-back Note: to speak between 2 containers in the same pod you don't need dns. Use localhost:port
kubeless	https://kubeless.io/docs/quick-start/ https://docs.bitnami.com/kubernetes/how-to/get-started-serverless-computing-kubeless/ https://github.com/kubeless/kubeless/releases
Debugging	https://kubernetes.io/docs/tasks/debug-application-cluster/debug-application/ https://sysdig.com/blog/debug-kubernetes-crashloopbackoff/ Kubectl describe pod {name} kubectl logs {pod}

Auto scale	**Horizontal Pod Autoscaler** watches and scales a pod in the deployment. https://www.tutorialspoint.com/kubernetes/kubernetes_autoscaling.htm kubectl autoscale deployment azure-vote-front --cpu-percent 50 --min 1 --max 10 kubectl get hpa while true; do curl http://192.168.99.105:30417; done
Package manager	https://medium.com/ingeniouslysimple/deploying-kubernetes-applications-with-helm-81c9c931f9d3 `sudo helm init` A **Chart** is a Helm package. helm create helloworld-chart helm repo update helm install stable/mysql helm inspect stable/mysql helm ls helm delete wintering-rodent
Monitoring	minikube addons enable heapster kubectl top node kubectl top pod kubectl get pods --all-namespaces minikube addons open heapster #liveness probe #readiness probe

Prometheus	Prometheus is an open source toolkit to monitor and alert helm init helm --tiller-namespace tiller version helm install stable/prometheus-operator --name prometheus-operator --namespace monitoring kubectl get pods -n monitoring Forward the Prometheus server to your machine so you can take a better look at the dashboard by opening http://localhost:9090 kubectl port-forward -n monitoring prometheus-prometheus-operator-prometheus-0 9090 http://localhost:9090/graph Prometheus has an expression browser for debugging purpose. To have a good-looking dashboard, use Grafana, it has a datasource ready to query on Prometheus. kubectl port-forward $(kubectl get pods --selector=app=grafana -n monitoring --output=jsonpath="{.items..metadata.name}") -n monitoring 3000 *The Alertmanager handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty, or OpsGenie. It also takes care of silencing and inhibition of alerts.* kubectl port-forward -n monitoring alertmanager-prometheus-operator-alertmanager-0 9093

Istio	A service mesh is a network of microservices that makeup applications and the interactions between them. https://istio.io/docs/examples/bookinfo/ https://github.com/istio/istio/tree/master/samples/bookinfo curl -L https://git.io/getLatestIstio | sh - cd istio-0.5.1 Open the file istio-1.0.2/install/kubernetes/istio-demo.yaml, search for LoadBalancer and replace it with NodePort. kubectl apply -f install/kubernetes/helm/istio/templates/crds.yaml kubectl apply -f install/kubernetes/istio-demo.yaml kubectl get service -n istio-system kubectl get pods -n istio-system https://thenewstack.io/tutorial-blue-green-deployments-with-kubernetes-and-istio/ https://medium.com/@bufferings/istio-bookinfo-on-minikube-1-deploy-app-7c953d18913d git clone https://github.com/redhat-developer-demos/istio-tutorial.git cd istio-tutorial git checkout book Netflix has recently announced it has stopped development of the Hystrix library in favor of the less well-known Resilience4J project. https://github.com/IBM/resilient-java-microservices-with-istio Azure https://blog.shanelee.name/2018/08/12/istio-on-azure-aks/ https://istio.io/docs/tasks/telemetry/using-istio-dashboard/ Circuit breaker https://istio.io/docs/tasks/traffic-management/circuit-breaking/
Patterns	https://github.com/gravitational/workshop/blob/master/k8sprod.md

Kubeadm, Kops	• Kubeadm is in the middle of the stack and it runs on each node, and basically creates and then talks to the Kubernetes API. • Kops on the other hand is responsible for the entire lifecycle of the cluster, from infrastructure provisioning to upgrading to deleting, and it knows about everything: nodes, masters, load balancers, cloud providers, monitoring, networking, logging etc.