**1 kubernetes overview , похожие проекты**

Kubernetes - это система оркестрации контейнеров, которая позволяет развертывать, масштабировать и управлять контейнерными приложениями.

Это не единственный контейнерный оркестратор! Несколько конкурентов Kubernetes:

Docker Swarm / Marathon / Amazon ECS / Azure Service Fabric / Hashicorp Nomad

Графические интерфейсы администратора: RedHat OpenShift  
  
**Kubernetes локально, какие есть варианты**   
https://rharshad.com/kubernetes-minikube-windows-setup/#running-kubernetes-cluster-in-virtualbox  
https://docs.docker.com/desktop/kubernetes/  
Процесс установки рассматривать не будем (показать в настройках enable + kubectl)

**Kubectl** The Kubernetes command-line tool, kubectl, allows you to run commands against Kubernetes clusters. You can use kubectl to deploy applications, inspect and manage cluster resources, and view logs.  
**(commands)** https://kubernetes.io/ru/docs/reference/kubectl/cheatsheet/  
 **Kubernetes Lens** is an effective, open-source IDE for Kubernetes. Lens simplifies working with Kubernetes by helping you manage and monitor clusters in real time.

**Компоненты Kubernetes**[**https://kubernetes.io/docs/concepts/overview/components/**](https://kubernetes.io/docs/concepts/overview/components/)

**The control plane** manages the worker nodes and the Pods in the cluster.

1. **etcd** — это распределенная база данных ключ-значение. Она является источником достоверных данных для Kubernetes. Каждый раз, когда вы вносите изменения в Kubernetes — например, отправляя YAML или JSON через REST API k8s или с помощью инструмента kubectl CLI (подробнее об этом позже) — эти изменения сохраняются в etcd в виде JSON.

2. **Сервер API** отвечает за обслуживание API Kubernetes.

3. **Менеджер контроллеров** — это мозг, который стоит за оркестрацией. Kubernetes имеет несколько контроллеров, каждый из которых отвечает за разные вещи**. Они следят за состоянием вашего кластера**, затем вносят или запрашивают изменения, когда это необходимо. (Node, Job, Endpoints controller)

4. **Scheduler** **Планировщик** распределяет работу между несколькими узлами. Он смотрит на требования к ресурсам (например, к процессору и памяти), чтобы определить, когда и на каком узле запускать под.

5. **Рабочие узлы. Воркеры содержат 2 основных компонента:**

**Kubelet** — это агент (небольшое приложение), который запускается на каждом рабочем узле в кластере. Его основная задача — **следить за тем, чтобы контейнеры запускались в поде** (обертке из одного или нескольких контейнеров). Но кто указывает ему запустить эти контейнеры? Это происходит из control plane (управляющего уровня) (где находится менеджер контроллеров). Когда уровень управления требует, чтобы что-то произошло в узле, kubelet делает это. **Менеджер контроллеров обращается к kubelet для управления подами.**

**Kube-proxy** обрабатывает сетевые соединения внутри и за пределами вашего кластера. Это означает, что если подам нужно общаться между собой или если какому-то внешнему сервису нужно обратиться к подам, kube-proxy поможет это сделать.

**Kubernetes resources: C:\kubernetes\lesson**

**Pod** - is the basic building block of kubernetes. A pod contains a group of one or more containers. Generally, each Pod has one container. (пример когда несколько контейнеров EFK stack fluentd fluentbit-внутри пода )

**Service** In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service). The set of Pods targeted by a Service is usually determined by a selector. A Service resource makes Pods accessible to other Pods or users outside the cluster. Without a Service, a Pod cannot be accessed at all. A Service forwards requests to a set of Pods: While the actual Pods that compose the backend set may change, the frontend clients should not need to be aware of that, nor should they need to keep track of the set of backends themselves.

https://kubernetes.io/docs/concepts/services-networking/service/

A **ReplicaSet**'s purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

The .spec.selector field is a [label selector](https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/). As discussed [earlier](https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/#how-a-replicaset-works) these are the labels used to identify potential Pods to acquire.

The .spec.template is a [pod template](https://kubernetes.io/docs/concepts/workloads/pods/#pod-templates)

https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

**A ReplicaSet ensures that a specified number of pod replicas are running at any given time.**

However, a **Deployment** is a higher-level concept that manages ReplicaSets and provides declarative updates to Pods along with a lot of other useful features.

**Volume**

At its core, a volume is a directory, possibly with some data in it, which is accessible to the containers in a pod.

To use a volume, specify the volumes to provide for the Pod in .spec.volumes and declare where to mount those volumes into containers in .spec.containers[\*].volumeMounts

A **PersistentVolumeClaim** (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources.

A **PersistentVolume** (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.

Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany

PersistentVolume: In this configuration, we instructed it to reserve 5GB of read-write storage at /mnt/data on the cluster’s node.

**WorkerNode** - physical machine -either virtual machine (like a computer server)

Что происходит

When you deploy applications on Kubernetes, you tell the control plane to start the application containers. The control plane schedules the containers to run on the cluster's nodes. The nodes communicate with the control plane using the Kubernetes API, which the control plane exposes. End users can also use the Kubernetes API directly to interact with the cluster.  
Когда мы деплоим приложения в кубернетес, мы обращаемся к control plane для

Ingress

https://kubernetes.io/docs/concepts/services-networking/ingress/

|  |  |
| --- | --- |
|  | Механизм, который обеспечивает маршрутизацию входящего трафика на уровне приложения |

https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-ingress-guide-nginx-example.html

Istio architecture

https://habr.com/ru/company/flant/blog/438426/

Istio перехватывает весь сетевой трафик и применяет к нему набор правил, вставляя в каждый pod умный прокси в виде sidecar-контейнера. Прокси, которые активируют все возможности, образуют собой **Data Plane**, и они могут динамически настраиваться с помощью **Control Plane**.

<https://istio.io/v1.4/docs/ops/deployment/architecture/>

Sidecar

Sidecar-контейнер — это контейнер, который должен быть запущен рядом с основным контейнером внутри пода. Этот паттерн нужен для расширения и улучшения функциональности основного приложения

<https://habr.com/ru/company/nixys/blog/559368/>

service mesh

Service mesh (буквально «сетка для сервисов») – это слой инфраструктуры, который позволяет вам управлять связью между микросервисами вашего ...

**Service Mesh** обычно реализуется путем предоставления каждому экземпляру сервиса экземпляра прокси, который называется Sidecar. Sidecar обрабатывают коммуникации между сервисами, производят мониторинг и устраняют проблемы безопасности, то есть все, что может быть абстрагировано от отдельных сервисов.

https://hsto.org/getpro/habr/post\_images/394/a6f/ca9/394a6fca966e4ef70ad205c6eb79af34.png