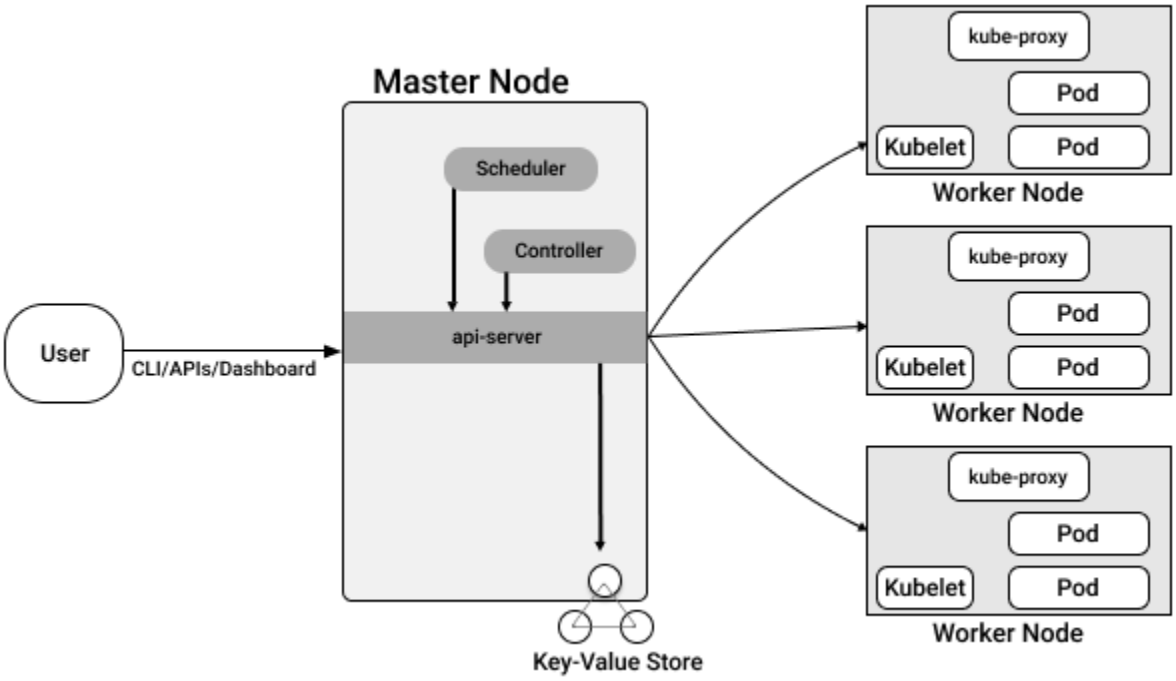
KUB 07: Kubernetes Control Plane

Описание:

В прошлых заданиях мы разобрались с установкой кластера, а так же с просмотром информации о нем с помощью kubectl. Вы так же уже знаете про такие компоненты кубернетес как etcd, api-server, scheduler и так далее. Наконец-то пришло время разобраться с ними поподробнее!
Если осознанно пойти на некоторые упрощения в пользу более прозрачной картины, то Kubernetes можно представить состоящим из трех базовых компонентов:

- один или более мастер-узел (Control Plane);
- один или более рабочий узел (Worker Node);
- распределенное key-value хранилище etcd.



Мастер-узел отвечает за функционирование кластера Kubernetes и является точкой входа для выполнения всех задач, связанных с кластером.
Взаимодействие с мастер-узлом происходит при помощи одного из выбранных вариантов:

- CLI (kubectl);
- GUI (Dashboard);
- API.

Для обеспечения отказоустойчивости рекомендуется использовать более одного мастер-узла. Для хранения состояния кластера мастер-узел работает с распределенным key-value хранилищем etcd.

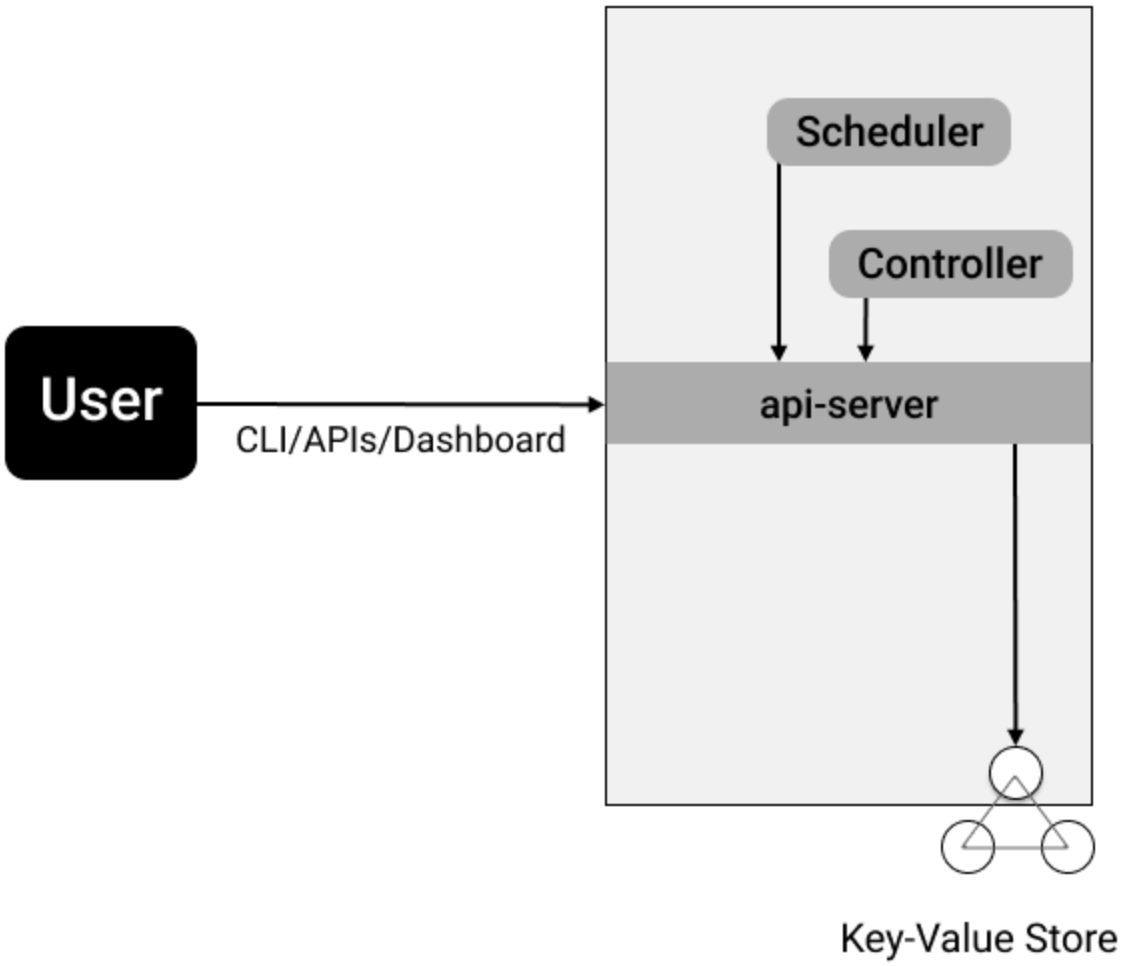
Мастер-узел (Control Plane) состоит из 4-х компонентов:

- API server
- Scheduler
- Controller-manager
- Etcd.

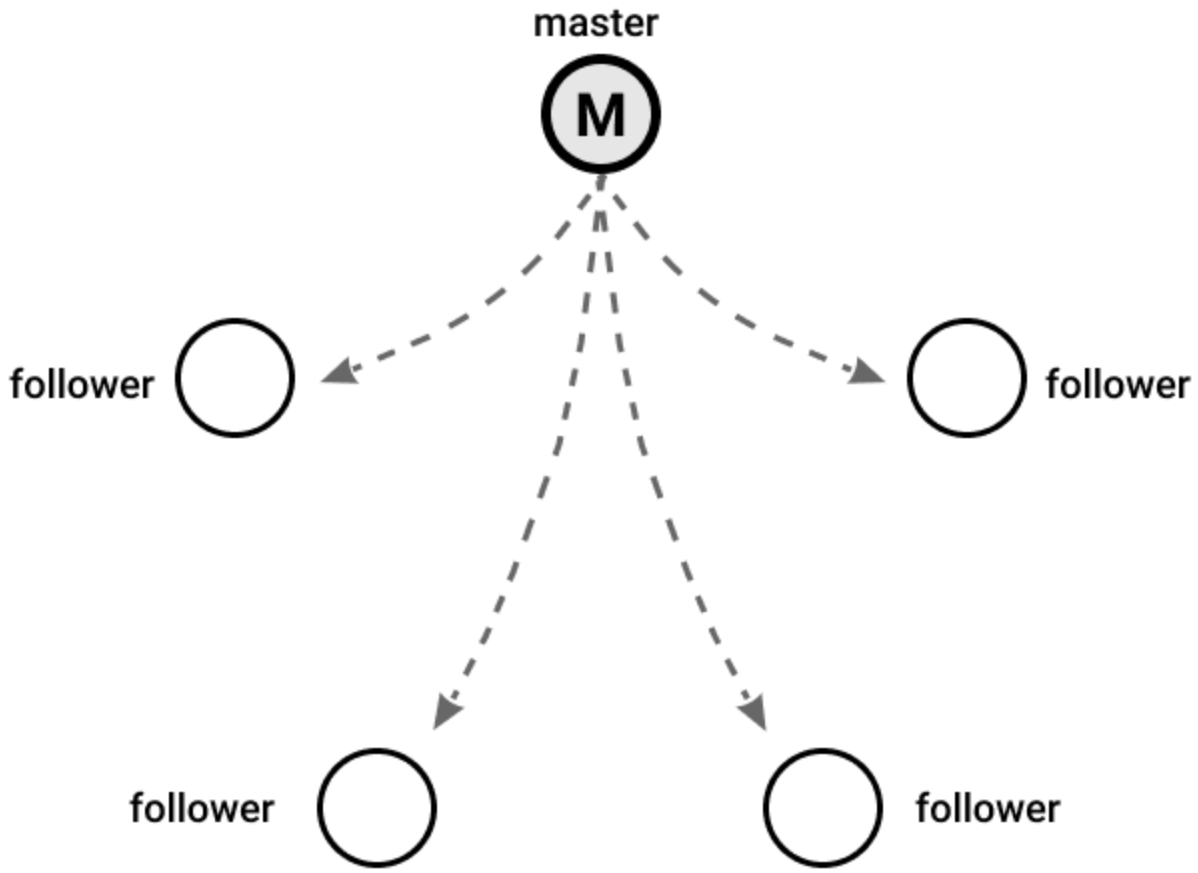
Вся работа с кластером происходит через API-сервер на мастер-узле посредством REST-запросов. К примеру, помните как мы вынимали информацию о запущенных подах в задании kubectl? Так вот - kubectl отправляла запрос на API сервер кубернетес, который принимал его и обращался в хранилище etcd, получал оттуда информацию и возвращал ее kubectl - то есть нам. Помимо клиентов типа kubectl, API сервер так же используется и рабочими нодами - kubectl, запущенных на них, подключается к API и проходит аутентификацию. После чего способен обращаться к API и забирать задачи на выполнение.

Так же к API подключаются такие сервисы как:

- scheduler распределяет задачи по узлам кластера. Когда вы создаете pod, scheduler получает это событие от API и назначает данному поду конкретный узел, на котором он должен запуститься. Узел выбирается на основании разных правил, которые мы будем изучать в 15 и 16 заданиях.
- controller-manager управляет состоянием объектов в кластере, следит за нынешним состоянием кластера через API, приводит нынешние состояние кластера к требуемому. Например, когда вы увеличиваете количество копий какого-то пода - именно controller-manager видит, что запущено сейчас 2 пода, а необходимо чтобы их было 5 и создает три дополнительных. После чего в дело вступает scheduler, который назначает этим подам конкретные ноды.
- etcd конечно не подключается к API, а API подключается к etcd, который хранит множество пар key-value и следит за их консистентностью. Практически любая информация, необходимая для работы кластера, содержится в etcd.



Etcd использует алгоритм Raft. Это позволяет пережить падение одного или нескольких узлов из группы и не только не потерять данные, но и сохранить их целостность. Один из участников кластера назначается мастером, остальные — ведомыми.



Kubernetes использует etcd для хранения состояния кластера, конфигурации сетей, configmaps, secrets. Кстати, в темах, которые нужны для прохождения сертификации CKA нужны etcd, так что мы рекомендуем изучить подробнее утилиту etcdctl.
Давайте попытаемся посмотреть список ключей, которые хранятся в etcd:

```
# etcdctl get --prefix --keys-only ''
/registry/apiextensions.k8s.io/customresourcedefinitions/bgpconfigurations.crd.projectcalico.org

/registry/apiextensions.k8s.io/customresourcedefinitions/bgppeers.crd.projectcalico.org

/registry/apiextensions.k8s.io/customresourcedefinitions/blockaffinities.crd.projectcalico.org

/registry/apiextensions.k8s.io/customresourcedefinitions/clusterinformations.crd.projectcalico.org

/registry/apiextensions.k8s.io/customresourcedefinitions/felixconfigurations.crd.projectcalico.org
```

```
... skipped ...
```

Как видно из вывода мы сразу попали на конфигурацию сетевого плагина calico.
Так же мы можем увидеть список всех участников etcd кластера:

```
# etcdctl member list
40d1e34f1ea6ae0b, started, etcd1, https://134.122.85.85:2380,
https://134.122.85.85:2379, false
```

В этом случае у нас всего лишь одна нода в кластере, что делает из кластера standalone версию.
Еще парочка полезных мелочей, которые могут вам оказаться полезны - это создание бекапа и его восстановление:

```
# etcdctl snapshot save /tmp/snap
{"level":"info","ts":1617228866.914506,"caller":"snapshot/v3_sna
pshot.go:119","msg":"created temporary db
file","path":"/tmp/snap.part"}
{"level":"info","ts":"2021-03-31T22:14:26.952Z","caller":"client
v3/maintenance.go:200","msg":"opened snapshot stream;
downloading"}
{"level":"info","ts":1617228866.9542065,"caller":"snapshot/v3_sn
apshot.go:127","msg":"fetching
snapshot","endpoint":"https://127.0.0.1:2379"}
{"level":"info","ts":"2021-03-31T22:14:28.430Z","caller":"client
v3/maintenance.go:208","msg":"completed snapshot read; closing"}
{"level":"info","ts":1617228868.454462,"caller":"snapshot/v3_sna
pshot.go:142","msg":"fetched
snapshot","endpoint":"https://127.0.0.1:2379","size":"16
MB","took":1.537897546}
{"level":"info","ts":1617228868.4559438,"caller":"snapshot/v3_sn
apshot.go:152","msg":"saved","path":"/tmp/snap"}
Snapshot saved at /tmp/snap
```

После чего можно посмотреть его статус и восстановиться в случае проблем:

```
# etcdctl snapshot status /tmp/snap
2be9c94d, 220042, 7898, 16 MB
# etcdctl snapshot restore /tmp/snap
{"level":"info","ts":1617228912.227521,"caller":"snapshot/v3_sna
pshot.go:296","msg":"restoring
snapshot","path":"/tmp/snap","wal-dir":"default.etcd/member/wal"
,"data-dir":"default.etcd","snap-dir":"default.etcd/member/snap"
}
```

```
{"level":"info","ts":1617228912.491522,"caller":"mvcc/kvstore.go
:380","msg":"restored last compact
revision","meta-bucket-name":"meta","meta-bucket-name-key":"fini
shedCompactRev","restored-compact-revision":212529}
{"level":"info","ts":1617228912.624972,"caller":"membership/clus
ter.go:392","msg":"added
member","cluster-id":"cdf818194e3a8c32","local-member-id":"0","a
dded-peer-id":"8e9e05c52164694d","added-peer-peer-urls":["http:/
/localhost:2380"]}
{"level":"info","ts":1617228912.6336265,"caller":"snapshot/v3_sn
apshot.go:309","msg":"restored
snapshot","path":"/tmp/snap","wal-dir":"default.etcd/member/wal"
,"data-dir":"default.etcd","snap-dir":"default.etcd/member/snap"
}
```


Итак, давайте подведем итоги:

1. В центре control plane находится API сервер, который принимает и обрабатывает запросы от других компонентов кубернетес или пользователей.
2. API сервер хранит свое состояние в распределенном хранилище etcd.
3. Scheduler - это отдельный сервис, который подключается к API и назначает новым подам хост, на котором они должны запускаться.
4. Controller Manager - так же отдельный сервис, который подключается к API и следит за объектами в кластере и изменяет их при наступлении каких-либо условий - например создает новые поды, если администратор увеличил их количество.
5. Прочие контроллеры - чуть позже мы поговорим с вами о том, как вы можете расширить Kubernetes кластер своим собственным API, но сейчас достаточно понимать что существует еще большой список контроллеров, которые вы можете установить в кубернетес и они будут так же подключаться к API и осуществлять какие-либо действия с ресурсами в kubernetes. Например, выдавать внешний IP адрес новым сервисам.

Полезные ссылки:

- Introduction to Kubernetes Architecture
- Operating etcd clusters for Kubernetes (official docs)

Задание:

1. Разверните кластер kubernetes на трех нодах, используя конфиг kubespray, подготовленный в 3ем задании:
- В control plane должны находиться ноды node1 и node2

- Etcd должен быть запущен на нодах node1, node2, node3
- Etcd должен быть запущен через docker (настройка по умолчанию в kubespray: inventory/kuber/group_vars/etcd.yaml: etcd_deployment_type: docker)

2. Сохраните бекап etcd на ноде node1 в /tmp/node1.etcd.backup
3. Остановите kube-apiserver на node2 (для этого надо удалить манифест /etc/kubernetes/manifests/kube-apiserver.yaml и рестартануть kubelet: systemctl restart kubelet).
4. Перенастройте kubectl на node2, чтобы он смотрел на ip адрес сервера node1.
5. Остановите etcd сервер на node2 (имеется ввиду остановить как и любой докер контейнер - используя docker stop)
6. Запустите под alpine с командой ping 8.8.8.8, используя kubectl на node2.
7. Отправьте задание на проверку.