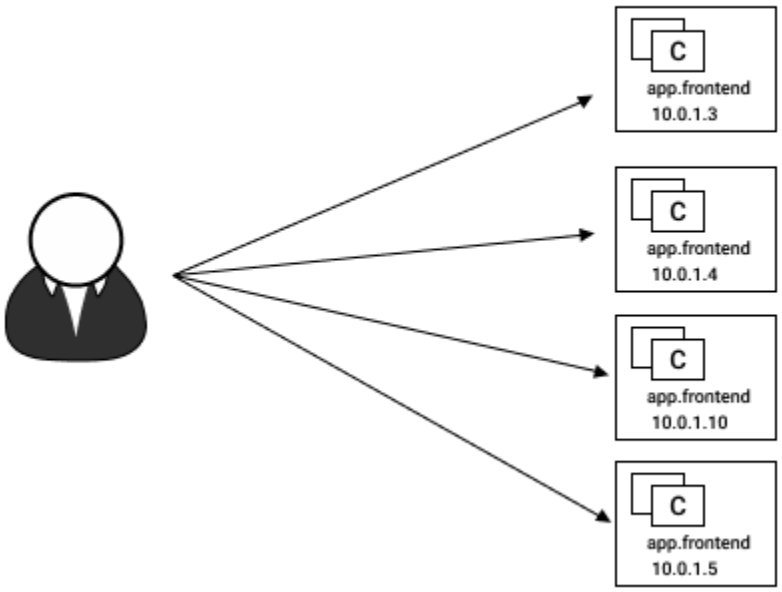
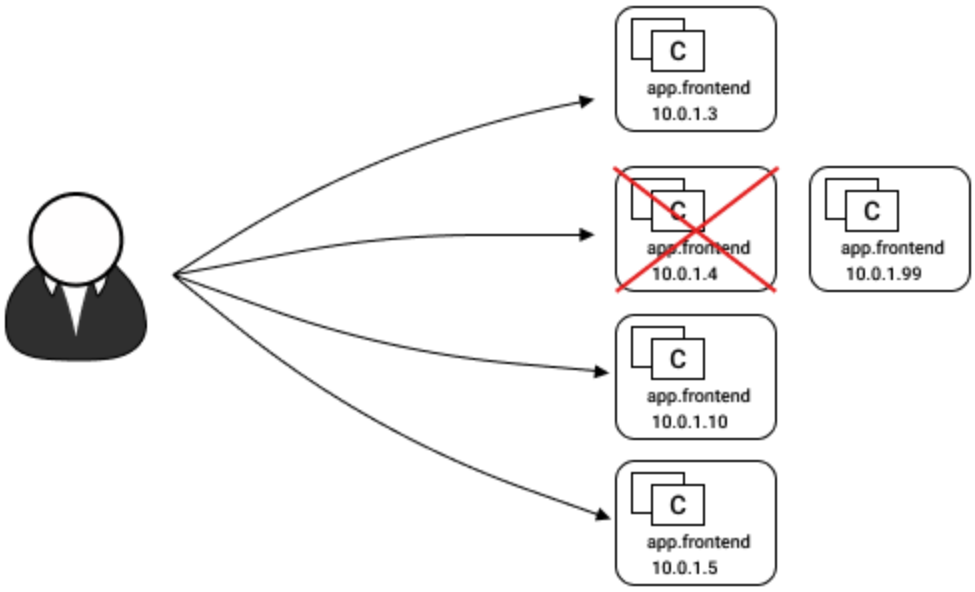
K8S 12: Kubernetes. Resources. Service

Описание:

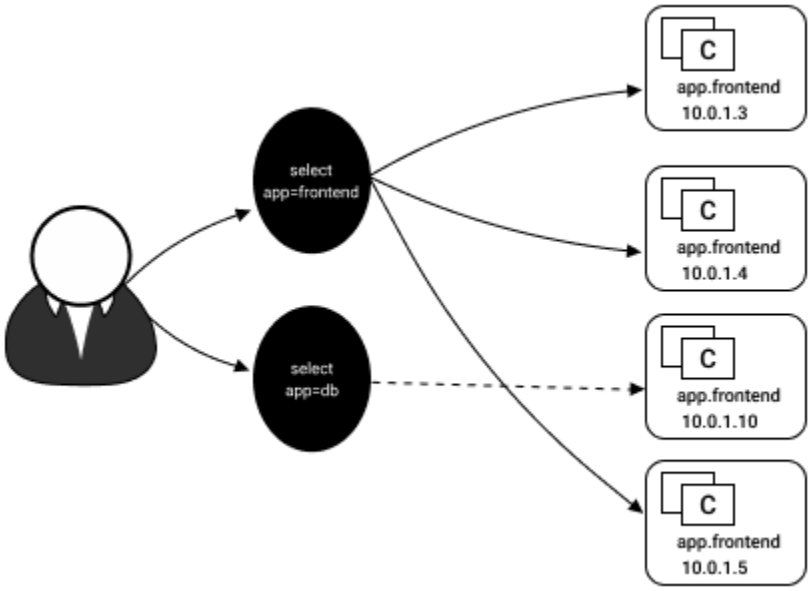
До этого момента мы не получали доступ к приложению внутри Pod. Данная абстракция позволит другим Pod или внешним пользователям получить доступ к приложениям. Представьте, что вы знаете IP каждого Pod и делаете curl http://$POD_IP



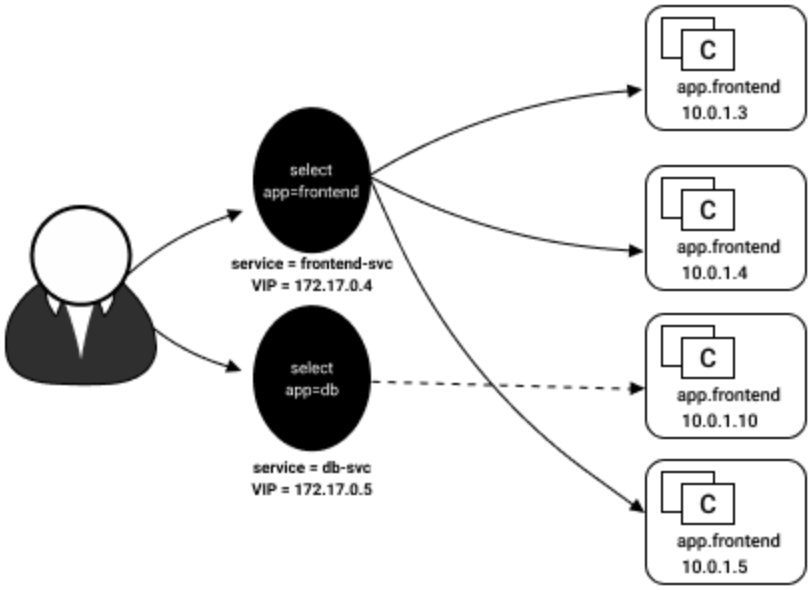
Все замечательно до момента падения одного из Pod, его тут же пересоздает ReplicaSet. Появляется новый Pod с новым IP. Опять нужно узнавать адрес Pod для получения к нему доступа.



Service позволяет обойти эту проблему. При помощи выборки можно сгруппировать Pod по меткам (labels). При замене Pod service, отслеживая изменения контейнеров с соответствующими метками, обновит информацию о том, куда должен быть направлен трафик, чтобы достучаться до функционирующей версии приложения.



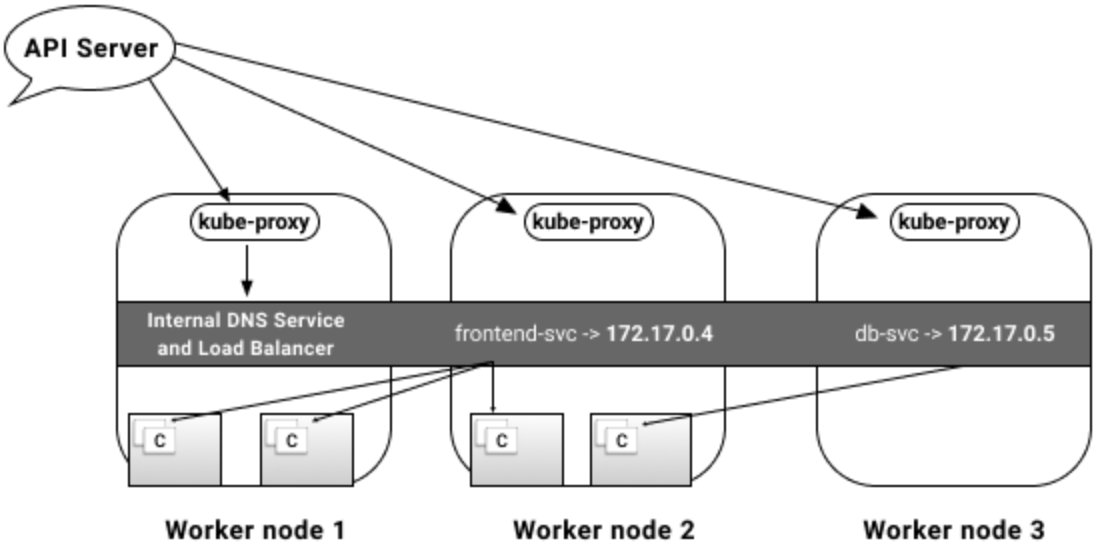
Это работает как простой round-robin балансировщик. Каждому объекту service выделяется свой IP.



Для создания сервиса достаточно указать метки и порт нужных Pod. Пример:

```yaml
kind: Service
apiVersion: v1
metadata:
  name: frontend-svc
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
```

После применения данного YAML будет создан объект service, но, кроме этого, kube-proxy также добавит на всех нодах правила в iptables для нового service. DNS-служба внутри кластера создаст имя для данного service.



Также после создания service в ENV контейнеров будут автоматически добавлены переменные с адресом service. Например, сервис redis-master добавит переменные:

```
REDIS_MASTER_SERVICE_HOST=172.17.0.6
REDIS_MASTER_SERVICE_PORT=6379
REDIS_MASTER_PORT=tcp://172.17.0.6:6379
REDIS_MASTER_PORT_6379_TCP=tcp://172.17.0.6:6379
REDIS_MASTER_PORT_6379_TCP_PROTO=tcp
REDIS_MASTER_PORT_6379_TCP_PORT=6379
REDIS_MASTER_PORT_6379_TCP_ADDR=172.17.0.6
```

Приложение может использовать переменные окружения для обнаружения redis-master.
DNS-имя для сервиса формируется по маске my-name.my-namespace.svc.cluster.local, где

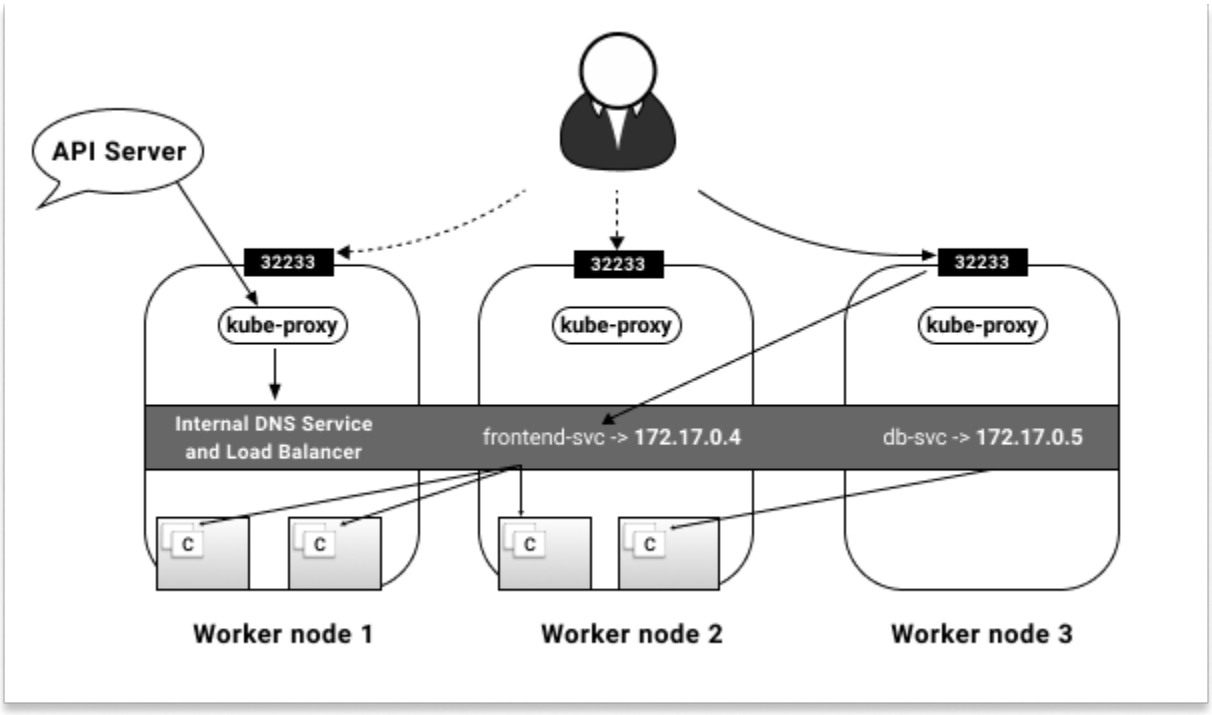
- my-name — имя сервиса (metadata.name);
- my-namespace — namespace, в котором создан сервис (metadata.namespace);
- svc.cluster.local — dns-суффикс кластера. Указывается при создании кластера.

ServiceType
При создании сервиса можно выбрать область доступности:

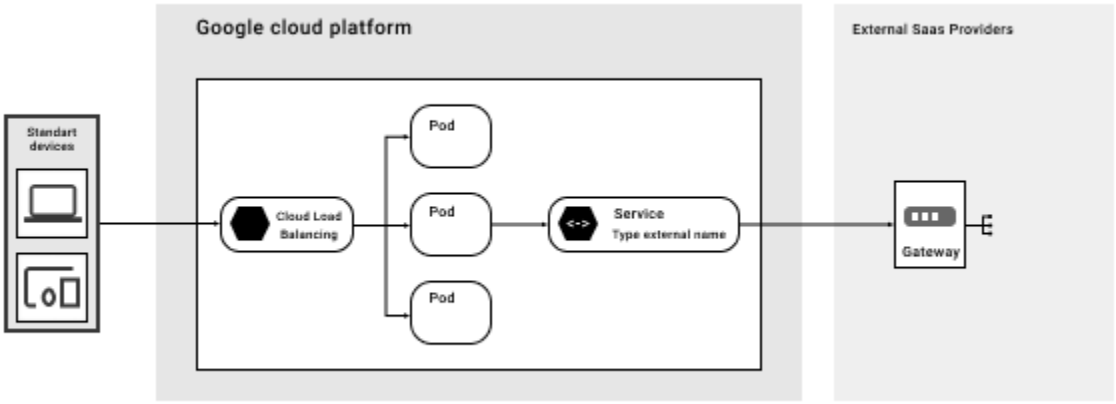
- доступен только внутри кластера — ClusterIP,
- доступен внутри кластера и для внешних клиентов — NodePort,
- предоставляет доступ к ресурсам вне кластера — ExternalService,
- балансировщик нагрузки — LoadBalancer, обычно необходима поддержка от облачного провайдера,
- внешний адрес — ExternalIP, требуется поддержка окружением/облаком.

Давайте разберем подробнее каждый из перечисленных вариантов.

ClusterIP
Данный тип используется по умолчанию при создании сервиса. Создается DNS-запись для сервиса и выделяется виртуальный IP.
NodePort
Похож на ClusterIP, но дополнительно выставляет открытый порт на каждом узле кластера. По умолчанию, порт назначается случайно из диапазона 30000-32767. После выделения порта kube-proxy прописывает правила в iptables, так что неважно, на какую ноду пришел пакет, он все равно дойдет до пода, даже если тот на другой ноде.

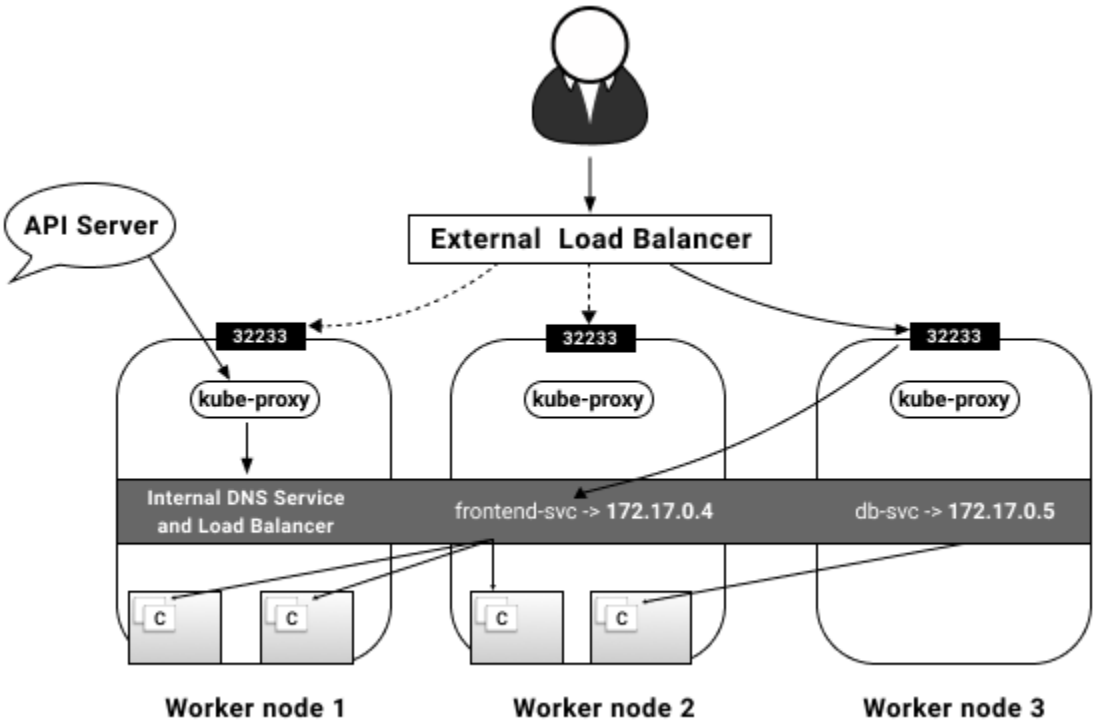


ExternalService
Есть несколько типов внешних сервисов. ExternalName связывает сервис с внешним DNS CNAME именем. Также возможно задать внешние IP-адреса в качестве endpoint сервиса.



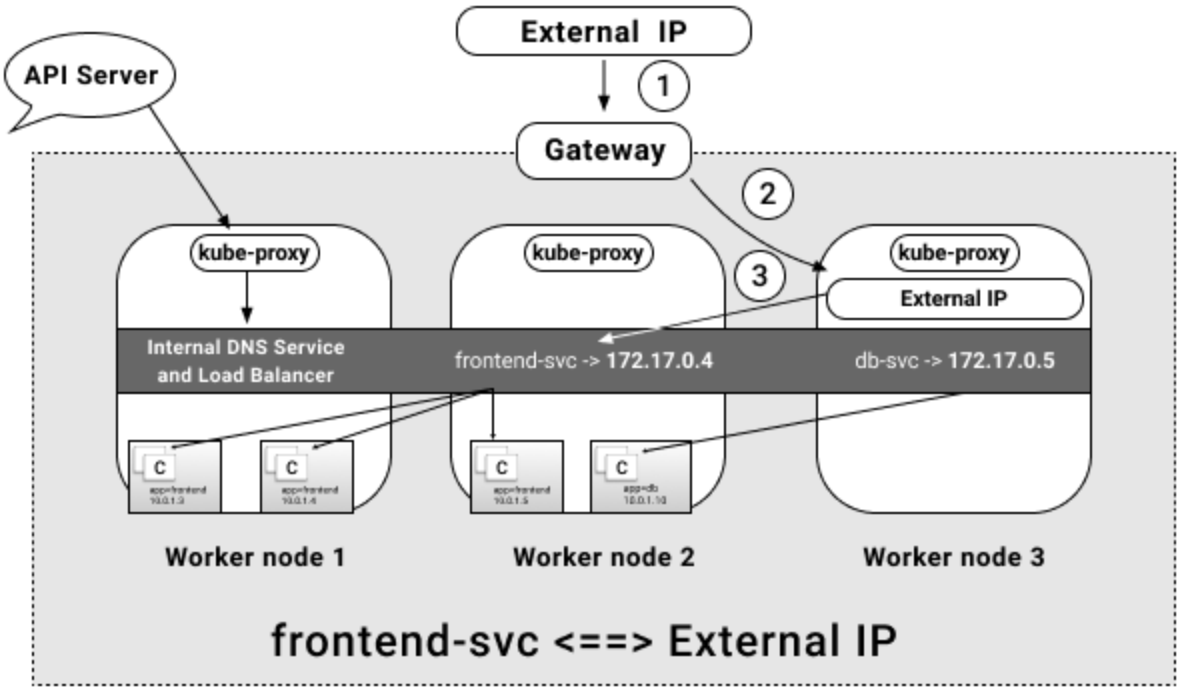
LoadBalancer

Данный тип сервиса часто зависит от поддержки окружения. Например, ELB на AWS. Как и nodeport, выставляет порты на каждом узле кластера, но дополнительно запускает внешний балансировщик нагрузки.



ExternalIP

Позволяет получить доступ к сервису с внешнего IP-адреса. То есть привязывает внешний IP-адрес на определенный сервис, почти как 1-to-1 NAT. Требуется поддержка окружением/облаком.



Каждый service содержит несколько адресов, на которые он балансирует запросы. Эти адреса называются endpoints. В большинстве случаев kubelet автоматически составляет список адресов на основе живых pod. Для externalservice адреса задаются статически. Пример:

```
kind: Service
apiVersion: v1
metadata:
 name: postgresql
spec:
 type: ClusterIP
 ports:
 - port: 5432
   targetPort: 543
---
kind: Endpoints
apiVersion: v1
metadata:
 name: postgresql
subsets:
 - addresses:
     - ip: 10.73.10.105
     - ip: 10.73.10.106
   ports:
```

```
    - port: 5432
```

Если pod будет обращаться к адресу postgresql, то kubelet автоматически направит трафик на указанные endpoints. Для pod это прозрачно, нет нужды записывать IP-адреса в конфигурацию.
Headless Service
Обычно каждому сервису выделяется виртуальный IP-адрес, при обращении к нему kube-proxy направит запрос на один из подов. Однако при создании сервиса можно указать None в поле clusterIP. В таком случае сервис не получит адреса, но по-прежнему получит список endpoints из подов. Такие сервисы используются для обнаружения всех подов в состоянии ready без использования виртуального адреса.
Также можно добавить аннотацию tolerate-unready-endpoints при создании сервиса и получить в списке endpoints все поды, даже не прошедшие readinescheck.

Полезные ссылки:

- Services in Kubernetes
- ClusterIP Service in Kubernetes
- NodePort Service in Kubernetes
- Mapping External Services
- Load Balancing Service in Kubernetes
- Service (official docs)
- Overview of a Service

Задание:

1. Создайте сервис для существующего деплоймента, используя примеры из описания выше. Сохраните получившуюся конфигурацию в first-deployment-service.yml.
2. Создайте сервис при помощи созданного в предыдущем пункте конфигурационного файла. Команду и вывод сохраните.
3. Проверьте состояние созданного сервиса. Команду и вывод сохраните.
4. Поменяйте тип service на NodePort и примените изменения. Команду и вывод сохраните.
5. Проверьте состояние созданного сервиса. Команду и вывод сохраните.
6. Проверьте состояние порта на ноде. Команду и вывод сохраните.
7. Пришлите в ответ все сохраненные команды и вышеуказанные выводы, а также файл first-deployment-service.yml.