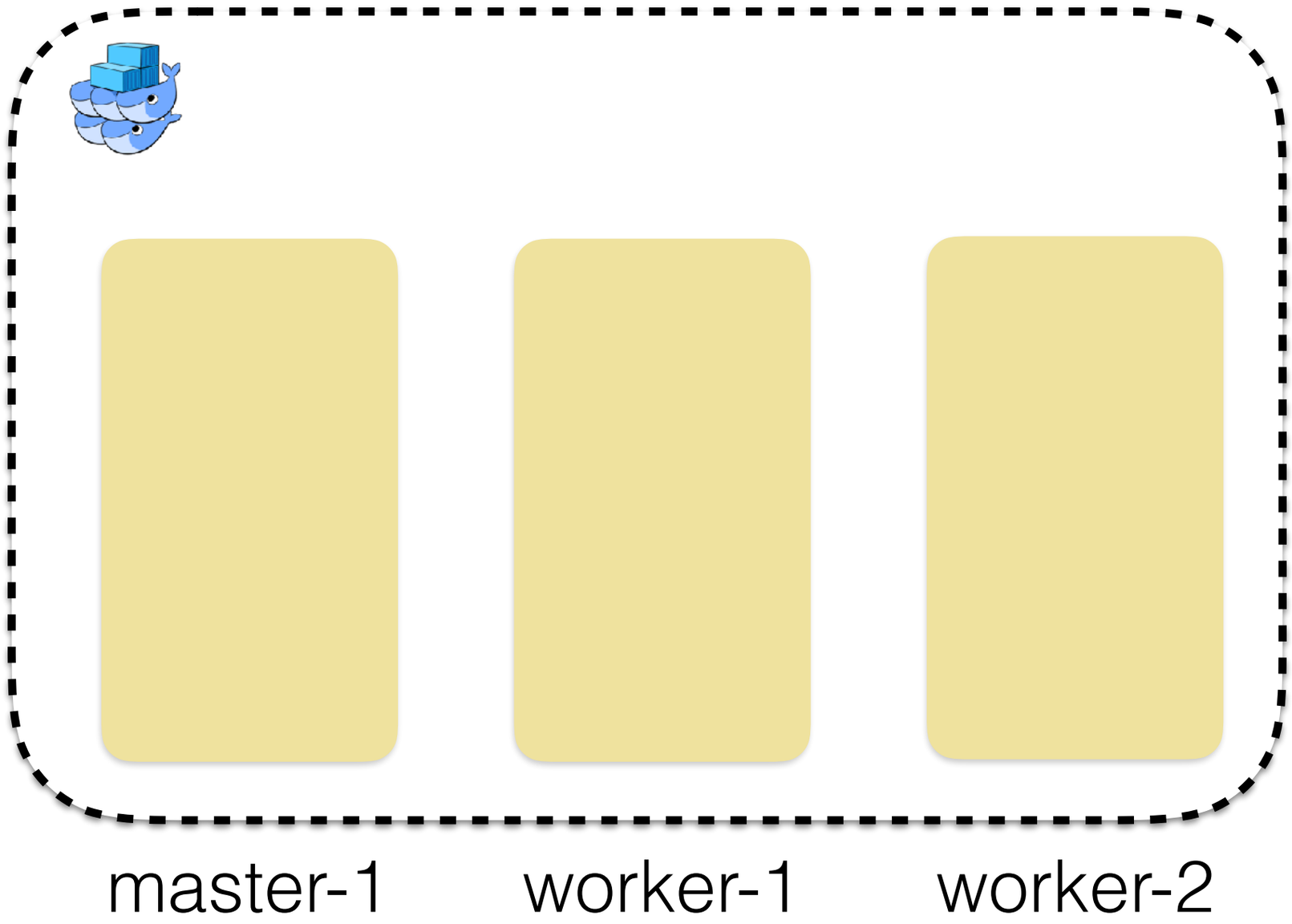
Docker swarm

Docker Swarm

- Поставляется из коробки с Docker (начиная с версии 1.12)
- Разворачивается 1-й командой (начиная с версии 1.12)
- Не использует внешние распределенные хранилища
- Позволяет описывать конфигурацию в формате Docker Compose
- Есть встроенный Service Discovery
- Есть встроенный Load Balancer

Swarm Cluster



Service

- Описывает что должно быть запущено и как (количество задач, необходимые ресурсы и т.д.)
- При запуске Service, *лидер* назначает *Worker*-ам необходимые задачи (*tasks*)
- Сервисы и их зависимости объединяем в Stack

Управление с помощью

- docker service create/update/rm/...
- docker stack deploy -c *compose-file.yml*

Stack

```
version: '3.3'
services:
  mongo:
    image: mongo:${MONGO_VERSION}
    volumes:
      - mongo_data:/data/db
    networks:
      back_net:
        aliases:
          - post_db
          - comment_db
post:
  image: ${USER_NAME}/post:${POST_VERSION}
  networks:
    - front_net
    - back_net

comment:
  image: ${USER_NAME}/comment:${COMMENT_VERSION}
  networks:
    - front_net
    - back_net
```

```
 ui:
    image: ${USER_NAME}/ui:${UI_VERSION}
    ports:
      - "${UI_PORT}:9292/tcp"
    networks:
      - front_net
volumes:
  mongo_data: {}

networks:
  back_net: {}
  front_net: {}
```

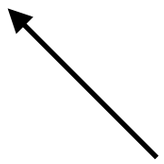
Stack

```
>> docker stack deploy  --compose-file docker-compose.yml ENV
```

не поддерживает переменные окружения и .env файлы

Workaround

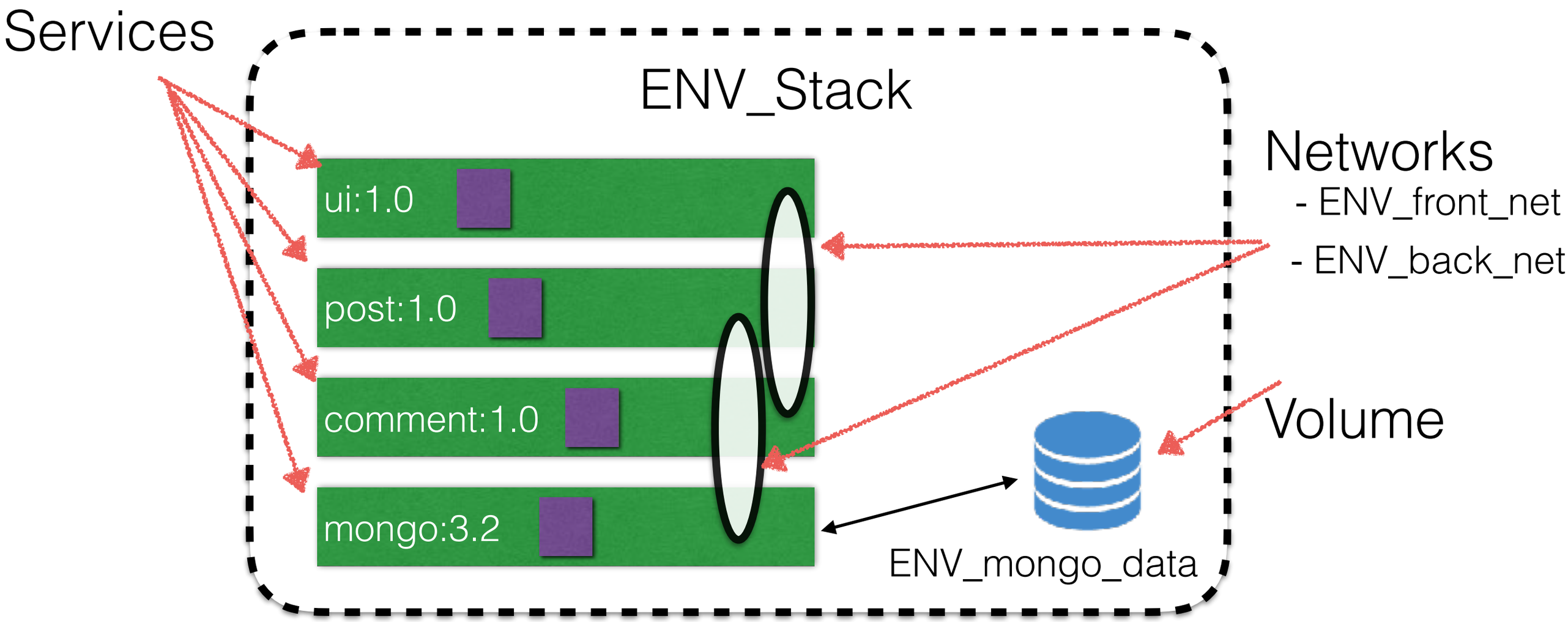
```
>> docker stack deploy --compose-file=<(docker-compose -f docker-compose.yml
config 2>/dev/null) ENV
```



Имя стека

Stack

>> docker stack deploy



Директивы планировщика

- Placement - **где разместить** задачу
- Deploy Mode - **как распределять** задачи
- Resources - **сколько ресурсов выделить** на задачу
- Update-policy - **как обновлять** задачи
- Restart-policy - **что делать**, если задача завершилась
- Labels - **какую метку присвоить** задаче
- Endpoint Mode - **как** задачам **взаимодействовать** между собой

Описание задачи

docker-compose.yml

```
post:
   image: ${USER_NAME}/post:${POST_VERSION}
   deploy:
     replicas: 2
     placement:
         constraints:
           - node.labels.worker == true
…
```

ИЛИ

```
>> docker service create …
```

Где разместить задачу?

```
$ docker service create \
  --name mongo \
  --constraint 'node.labels.reliability == high’  \
    --constraint 'engine.labels.provider == google’  \
  mongo:3.2
```

```
mongo:
  image: mongo:3.2
  deploy:
    placement:
      constraints:
        - node.labels.reliability == high
        - engine.labels.provider == google
…
```

2 типа меток:

- node
- engine

Labels

добавить node label

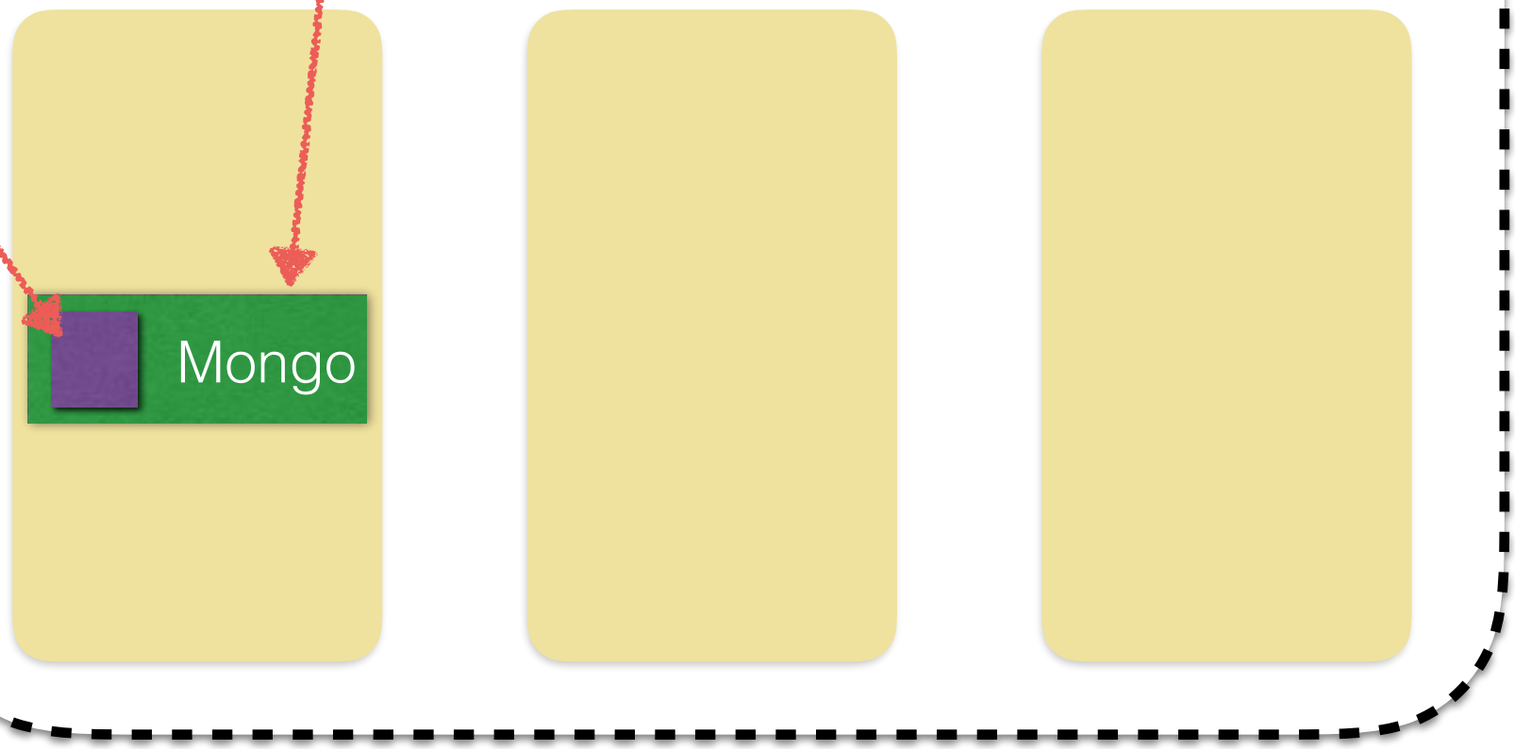
```
>> docker node update --label-add reliability=high master-1
```

```
>> /usr/bin/dockerd --label provider=google …
```

добавить engine label

Сервис

Контейнер



master-1 worker-1 worker-2

reliability=high

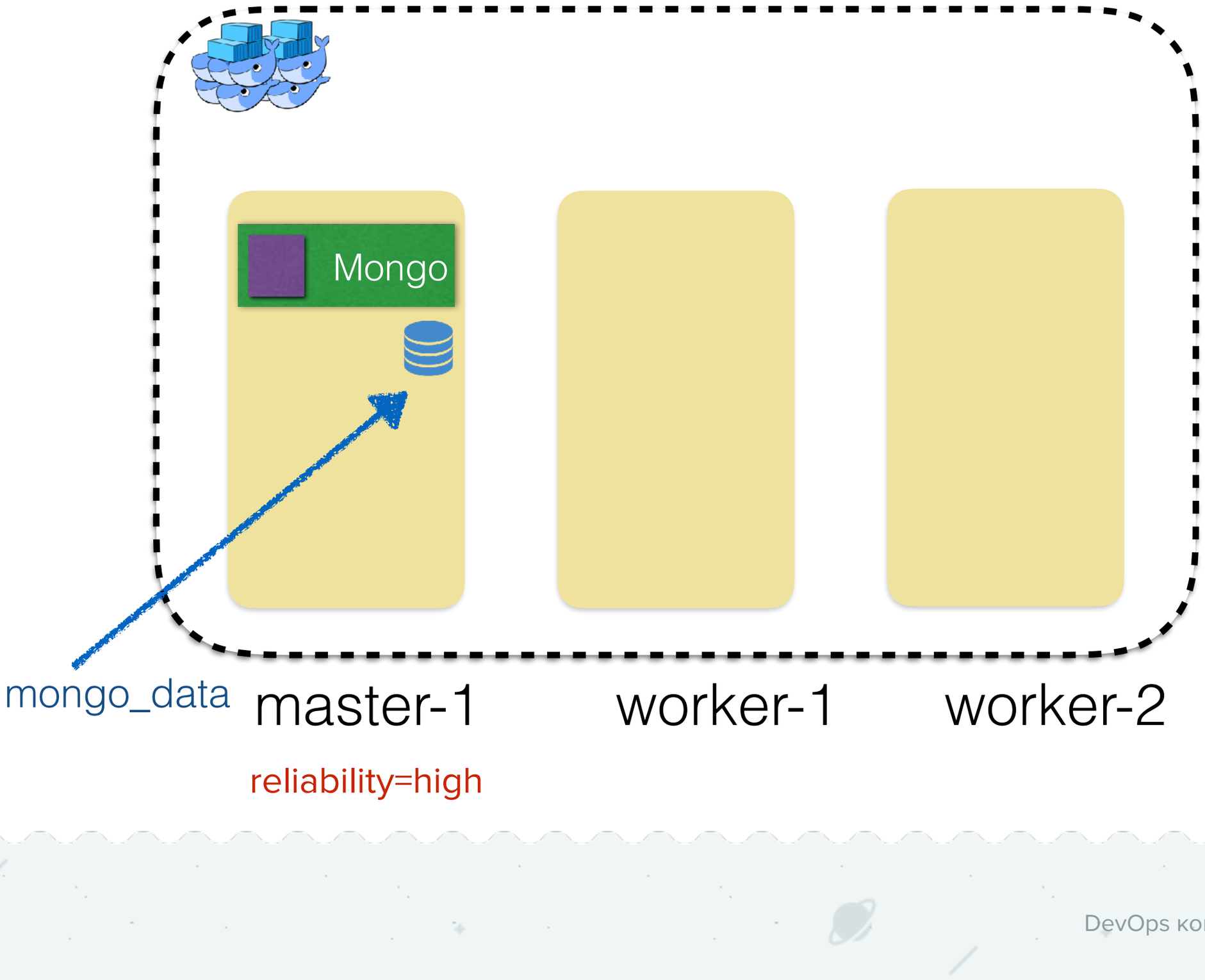
А что с данными?

```
mongo:
    image: mongo:${MONGO_VERSION}
    deploy:
      placement:
        constraints:
          - node.labels.reliability == high
          - engine.labels.provider == google
    volumes:
      - mongo_data:/data/db
```

Экземпляр данных будет доступен только на одной машине

Как думаете чем это может грозить?

Volume


Mongo
mongo_data
master-1
worker-1
worker-2
reliability=high

Deploy Mode

Как распределять задачи (в соответствии с placement) ?

- replicated - запустить определенное число задач (default)
- global - запустить задачу на каждой ноде

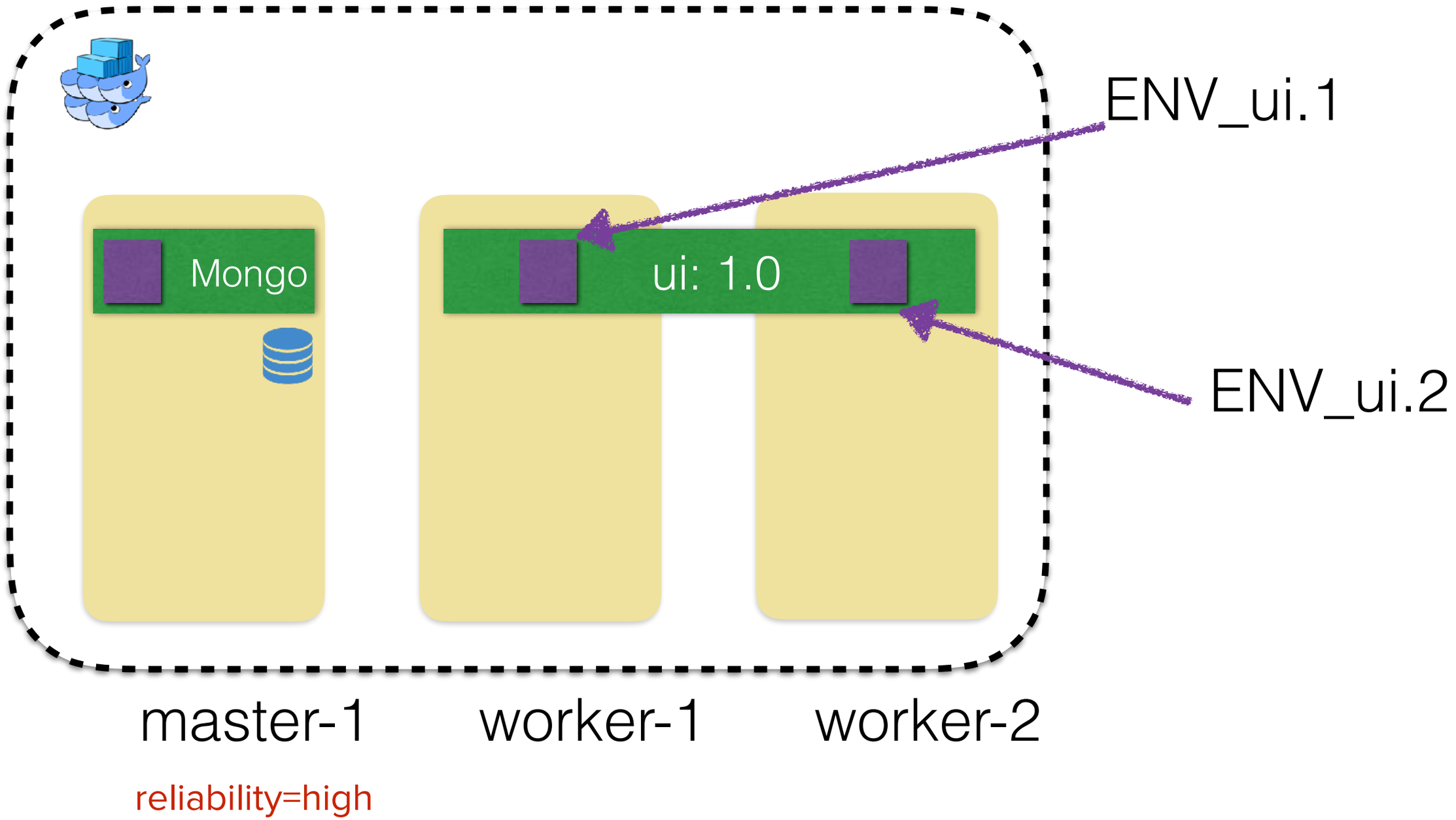
Replicated

Replicated - запустить определенное число задач (default)

```
ui:
  image: ${USER_NAME}/ui:${UI_VERSION}
  deploy:
    mode: replicated
    replicas: 2
    placement:
      constraints:
        - node.role == worker
…
```

Не будем размещать на manager

Replicated


ENV_ui.1
Mongo
ui: 1.0
ENV_ui.2
master-1
worker-1
worker-2
reliability=high

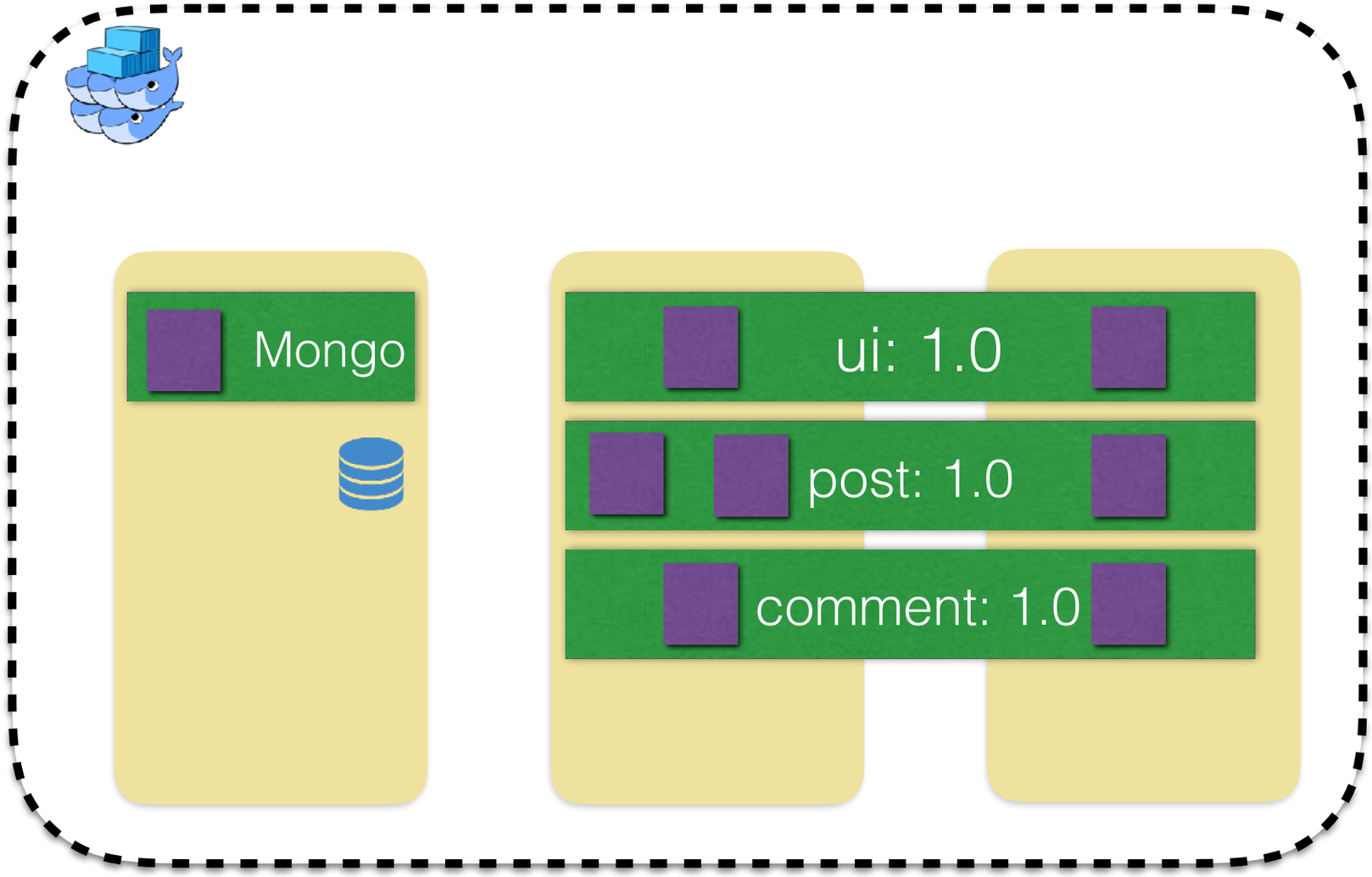
Replicated

Запустим остальные микросервисы

```
post:
  image: ${USER_NAME}/post:${POST_VERSION}
  deploy:
    mode: replicated
    replicas: 3
    placement:
     …

comment:
  image: ${USER_NAME}/comment:${COMMENT_VERSION}
  deploy:
    mode: replicated
    replicas: 2
    placement:
…
```

Replicated



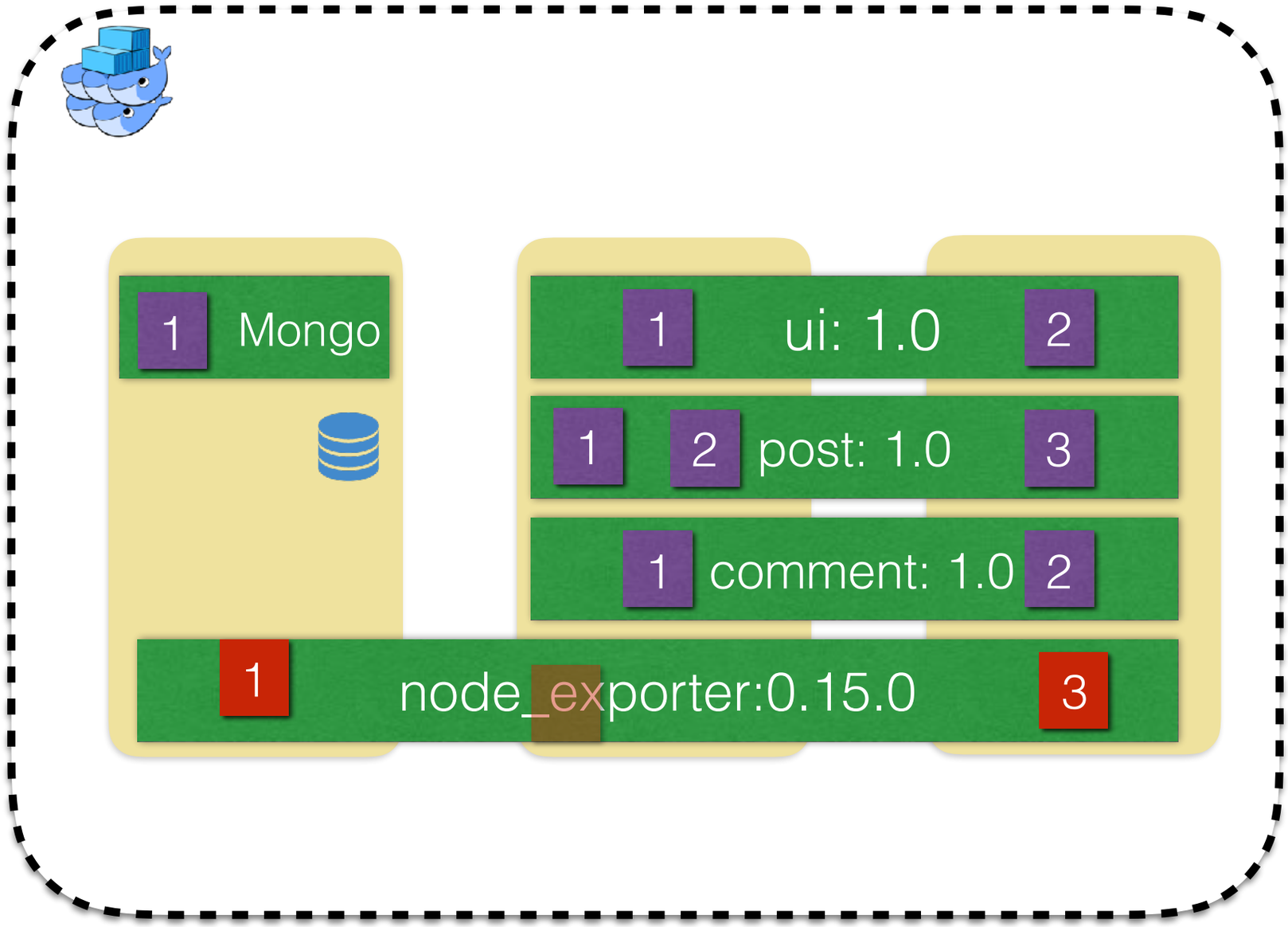
reliability=high

Global

Global - запустить на задачу на каждой ноде

```
node-exporter:
    image: prom/node-exporter:v0.15.0
    deploy:
      mode: global
    user: root
    volumes:
      - /proc:/host/proc:ro
    …
```

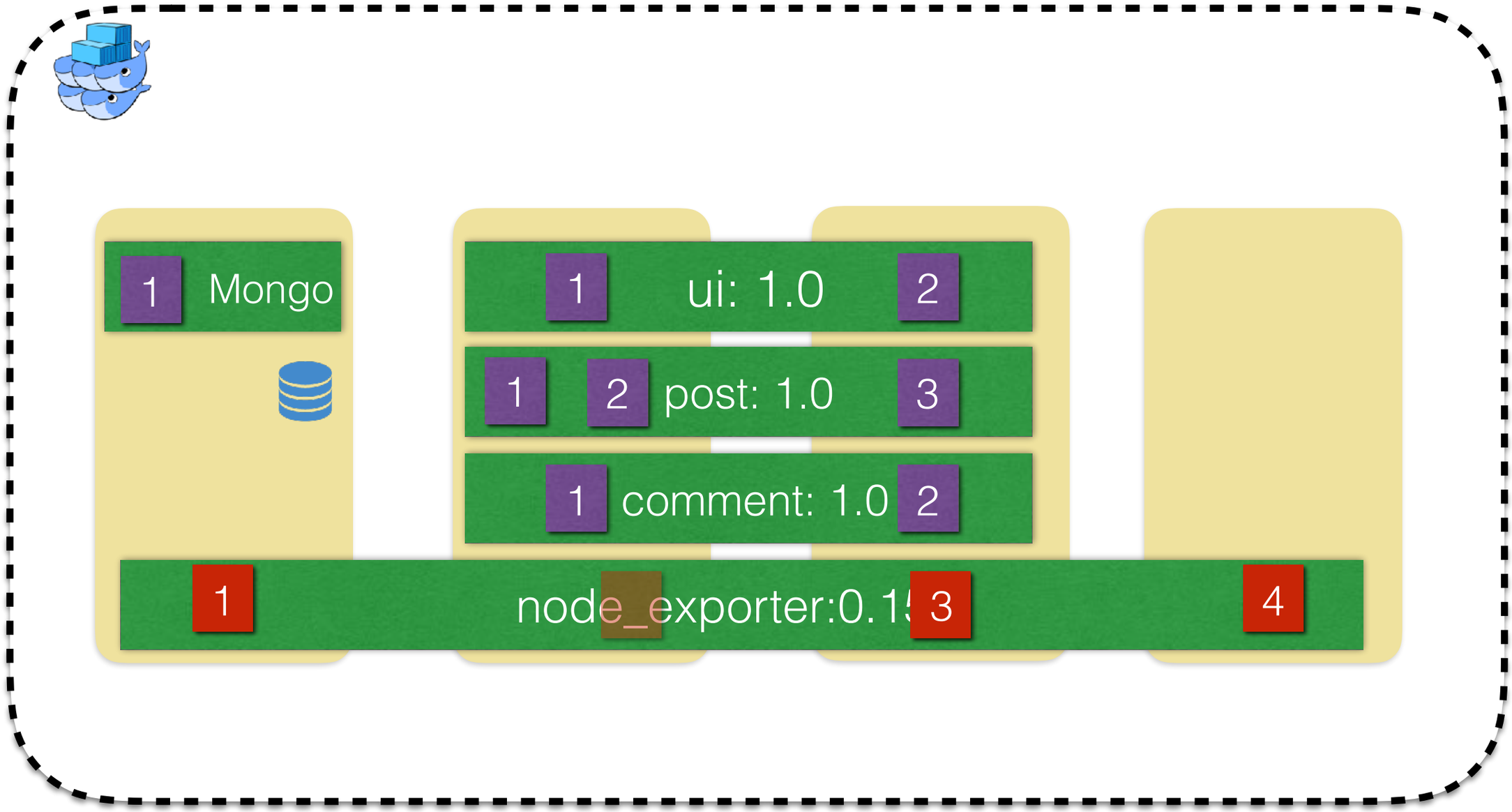
Global



master-1 worker-1 worker-2

reliability=high

Global

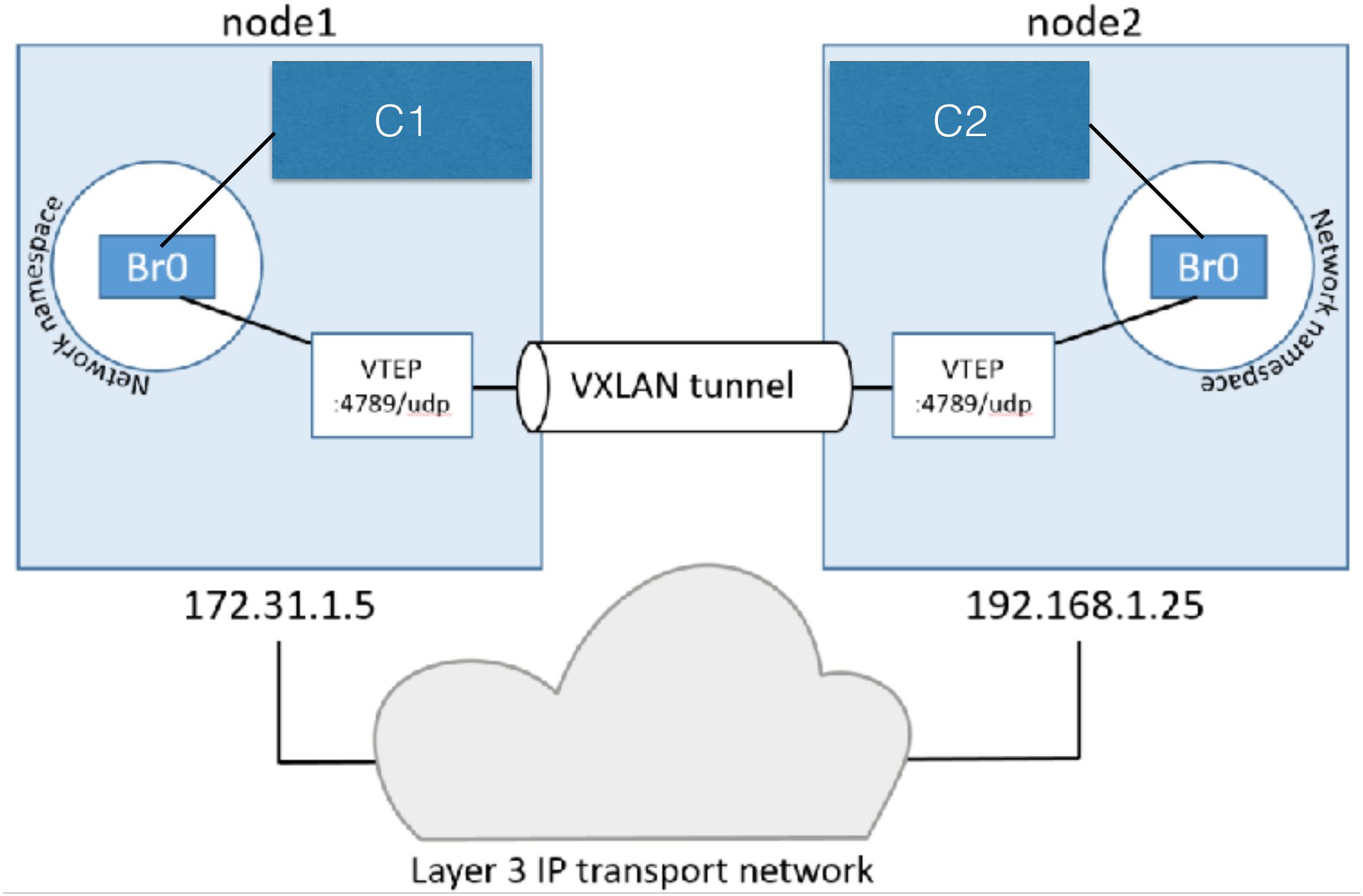


reliability=high

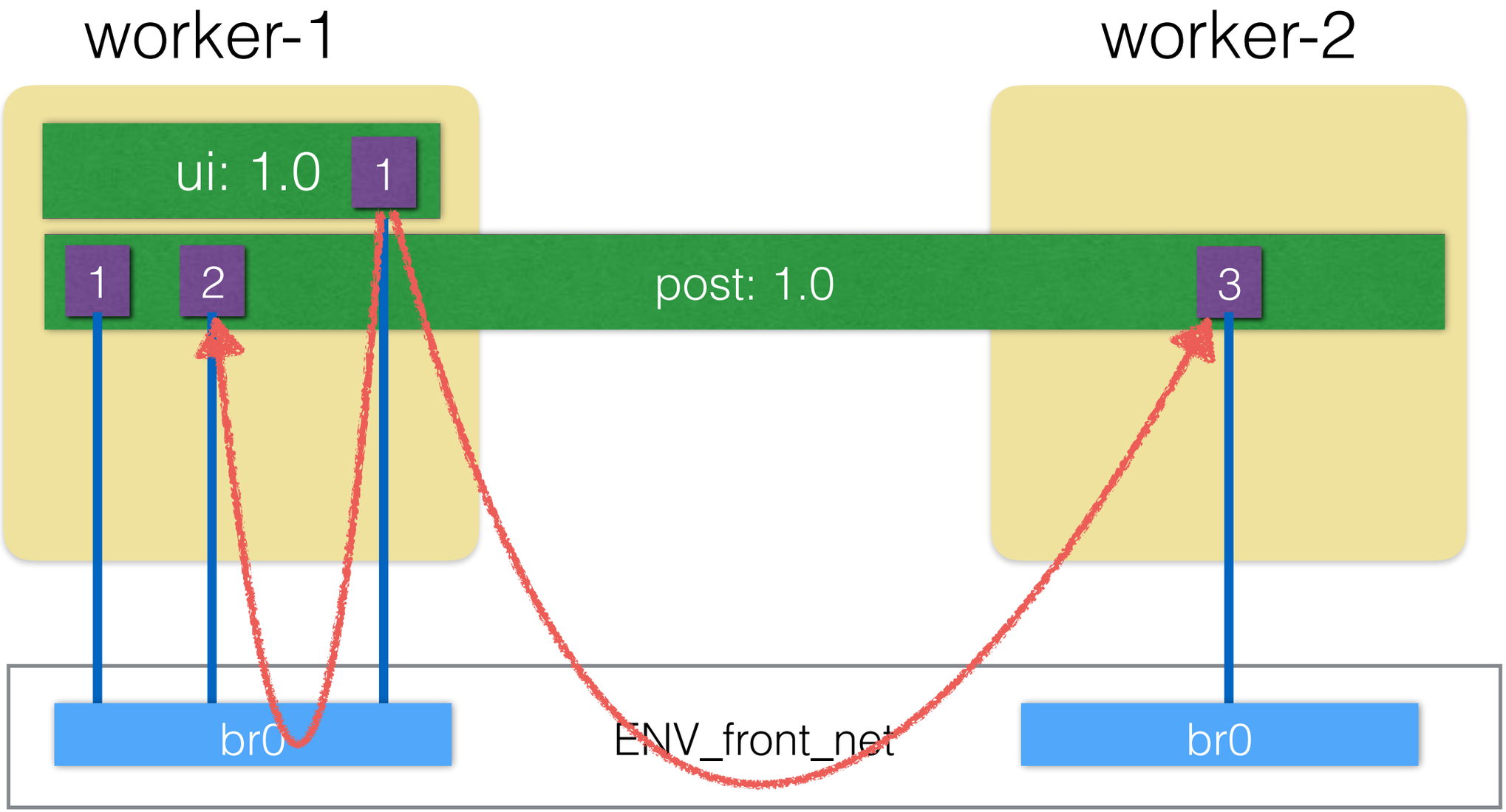
Overlay network

- Overlay network – позволяет объединить в одну сеть контейнеры нескольких докер хостов.
- работает поверх vxlan
- Нужно хранить состояние распределенной сети

Overlay network



Как им общаться между собой?

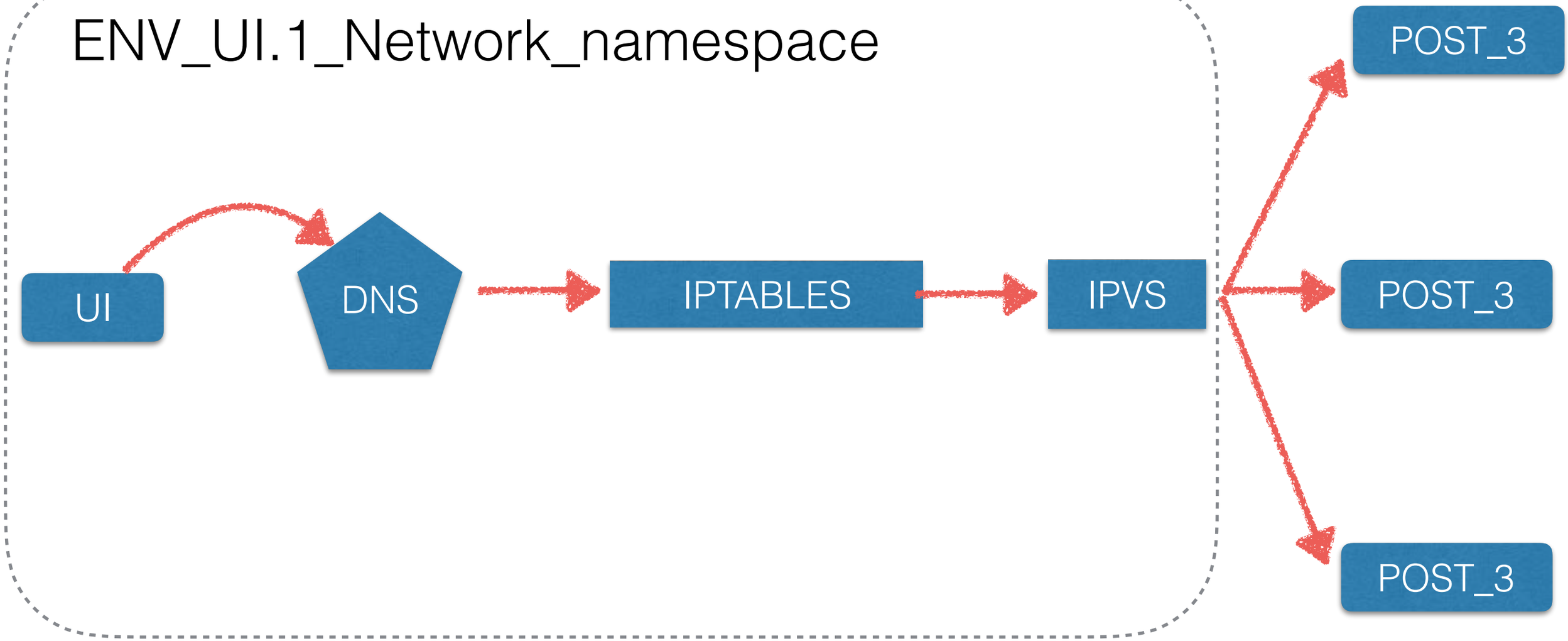


IP Virtual Server (IPVS)

- Реализация балансировщика нагрузки в ядре Linux
- Включен по-умолчанию (Round Robin LB)
- Сервису назначается 1 VIP (Виртуальный IP-адрес)
- Клиенты не знают, сколько за этим адресом участников

```
post:
  image: ${USER_NAME}/post:${POST_VERSION}
  deploy:
    endpoint_mode: vip
    replicas: 3
```

IP Virtual Server (IPVS)



```
>> ip netns exec ca4f571abfe2 ipvsadm
FWM  305 rr
  -> 10.0.0.9:0
  -> 10.0.0.22:0
  -> 10.0.0.23:0
```

3: dst: 10.0.0.8

1: nslookup **post**

UI

DNS

IPTABLES

IPVS

POST_1

POST_2

POST_3

6: dst: 10.0.0.8
+ mark 0x131
(=305)

7: dst: 10.0.0.9

8: dst: 10.0.0.22

9: dst: 10.0.0.23

2:

Server: **127.0.0.11**
Non-authoritative answer:
Name: post
Address: 10.0.0.8

4: dst: 10.0.0.8

5: dst: 10.0.0.8
+ mark 0x131 (=305)

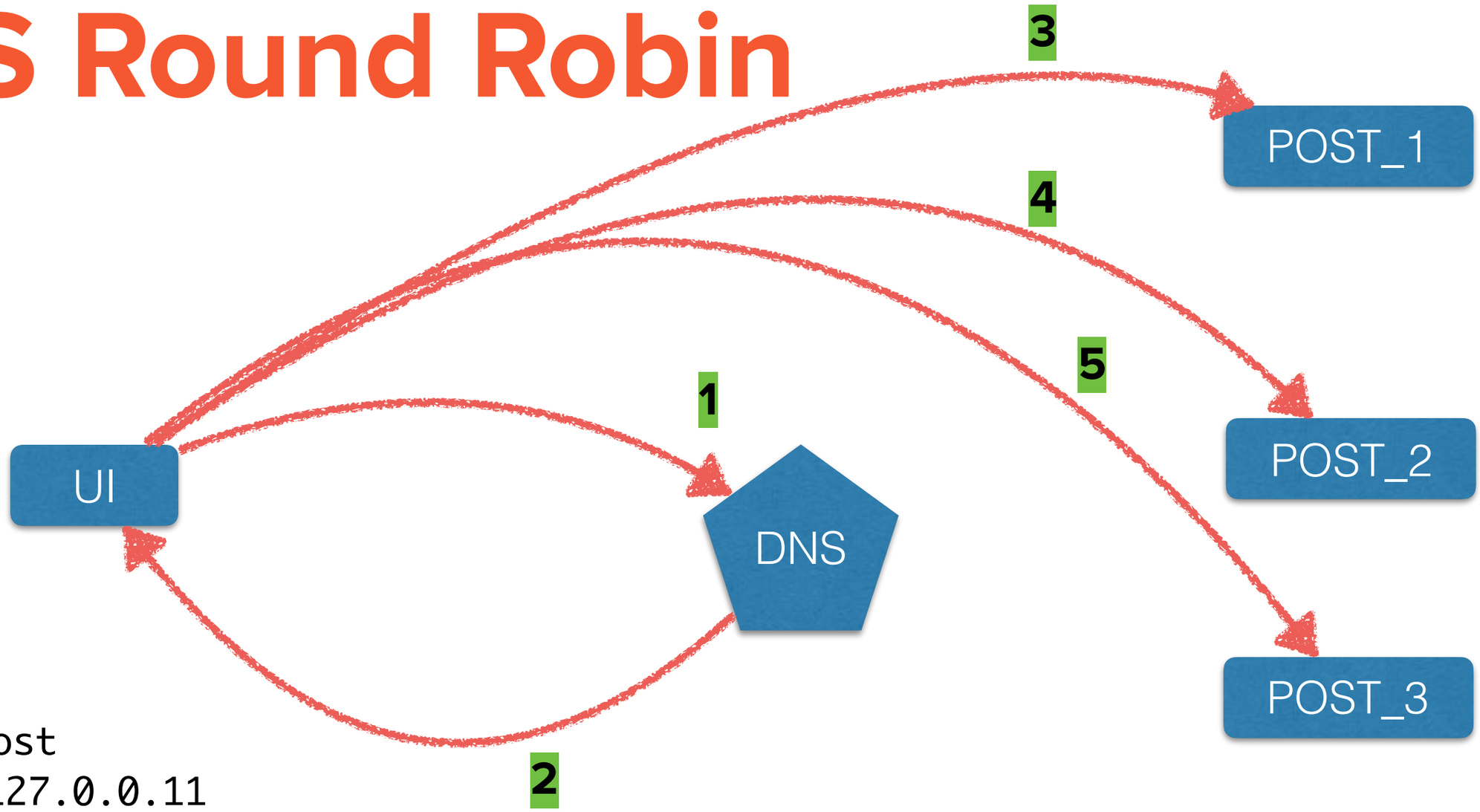
```
>> ip netns exec ca4f571abfe2 iptables -L -t mangle
…
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
MARK       all  --  anywhere             10.0.0.8             MARK set 0x131
```

DNS Round Robin

Включается отдельно.
DNS возвращает адреса всех участников

```
  post:
    image: ${USER_NAME}/post:${POST_VERSION}
    deploy:
      endpoint_mode: dnsrr
      replicas: 3
```

DNS Round Robin



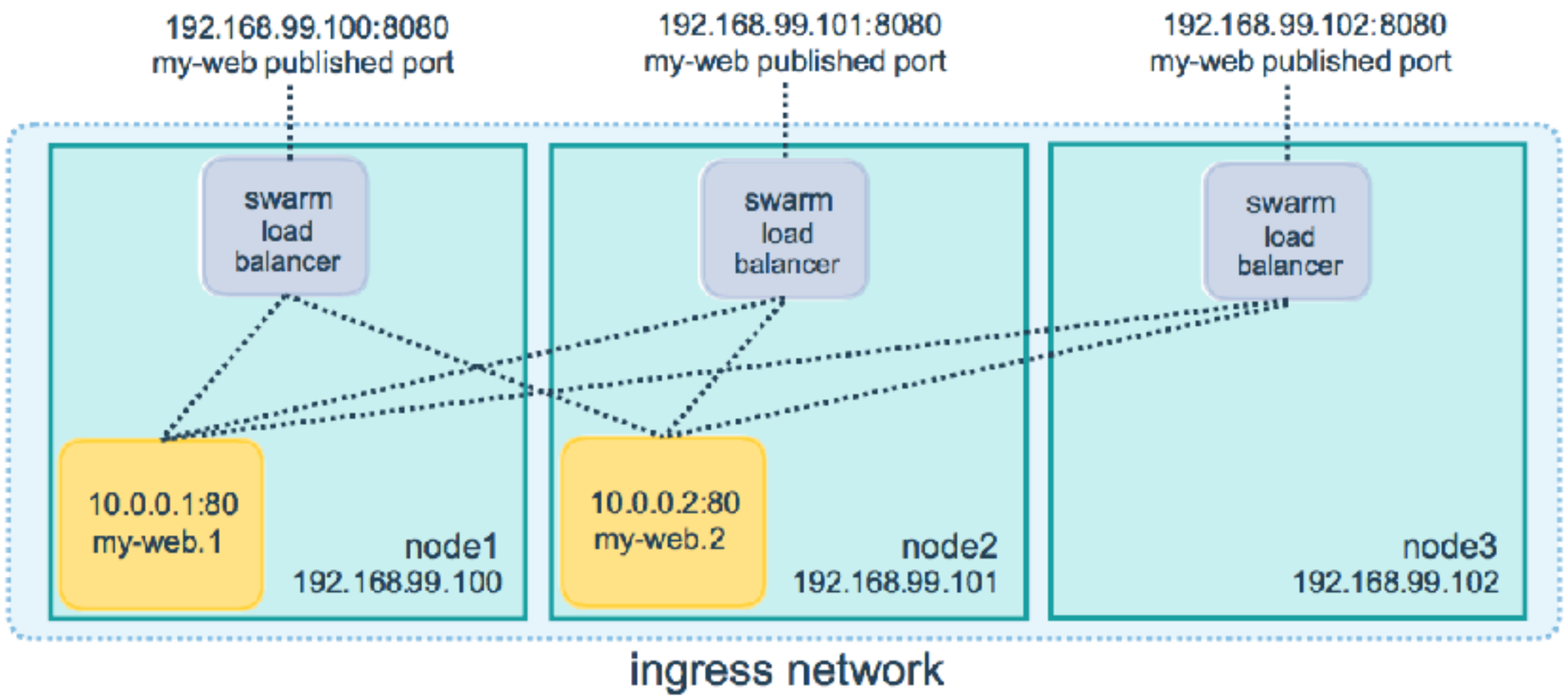

```
>>nslookup post
Server:     127.0.0.11
Address:  127.0.0.11#53

Non-authoritative answer:
Name:  post
Address: 10.0.0.29
Name:  post
Address: 10.0.0.28
Name:  post
Address: 10.0.0.27
```

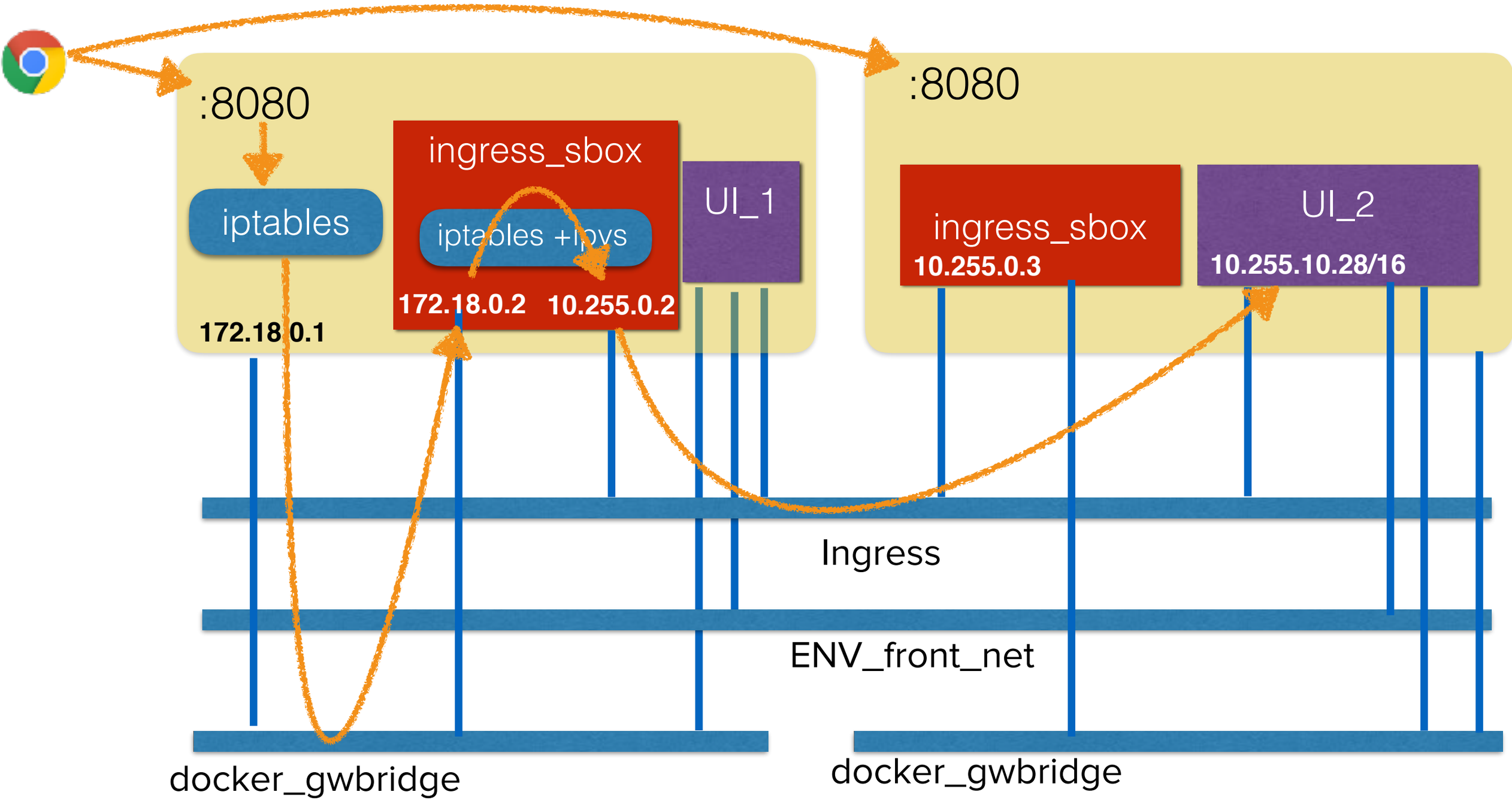
Как мы общаемся с приложением?

```
ui:
   image: ${USER_NAME}/ui:${UI_VERSION}
   deploy:
     …
   ports:
     - “8080:9292/tcp”
```

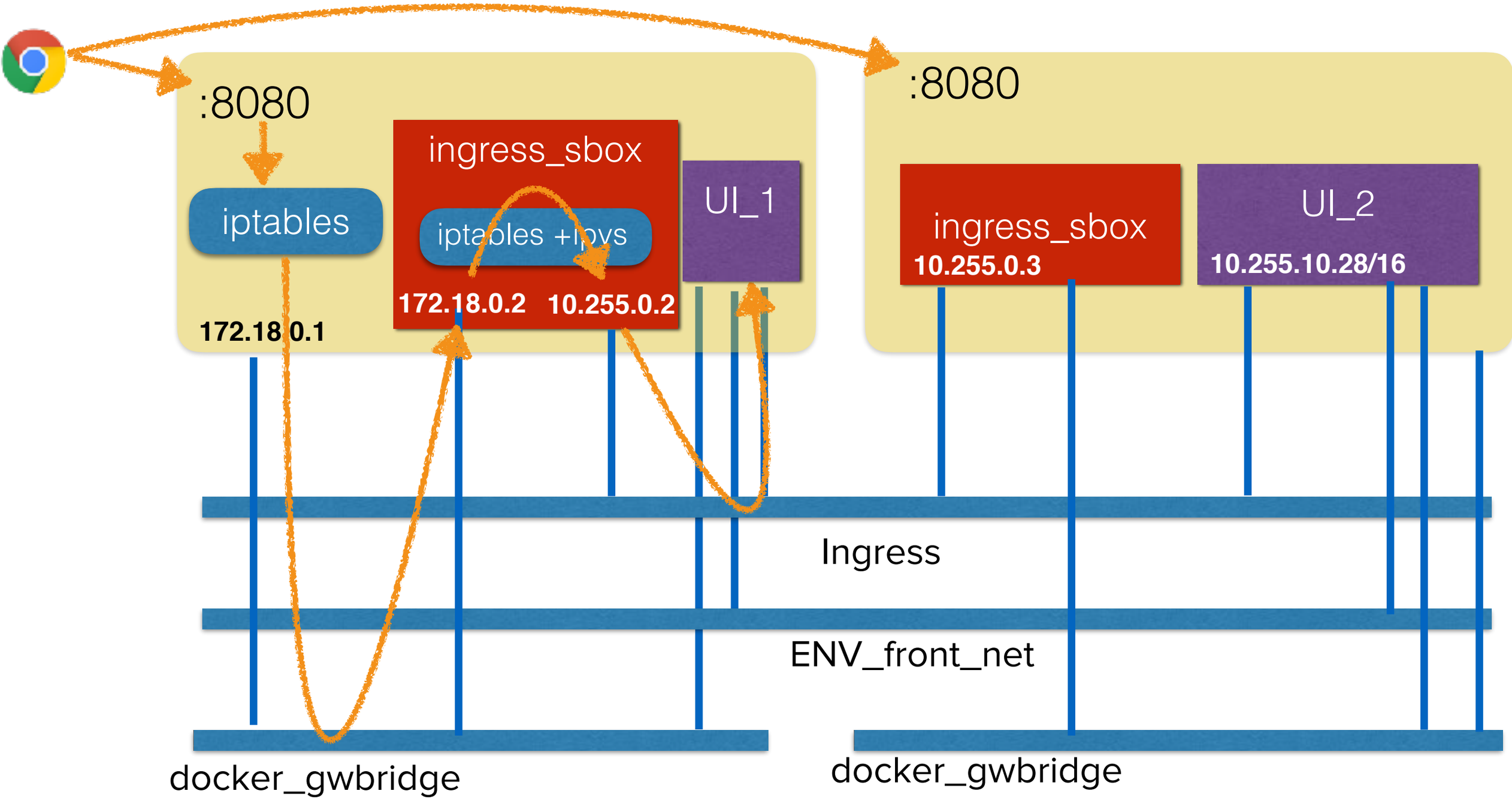
Routing mesh



Routing mesh



Routing mesh

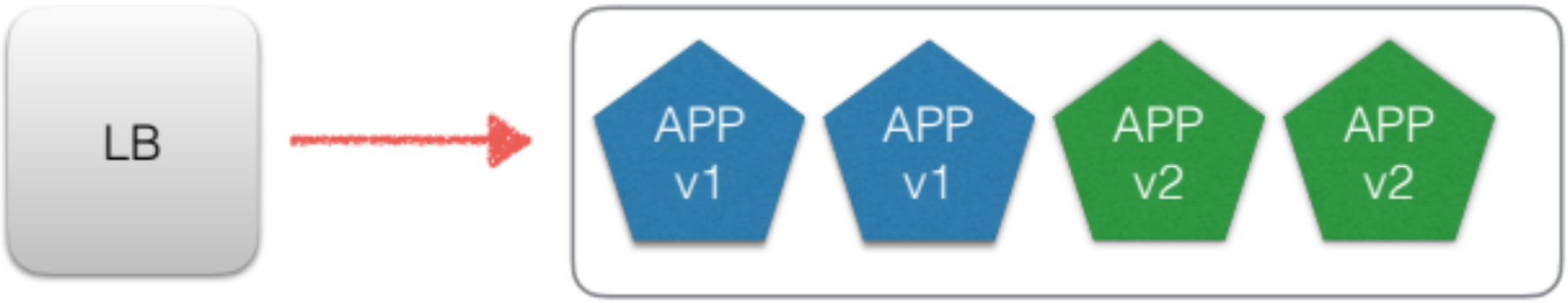


Update Service

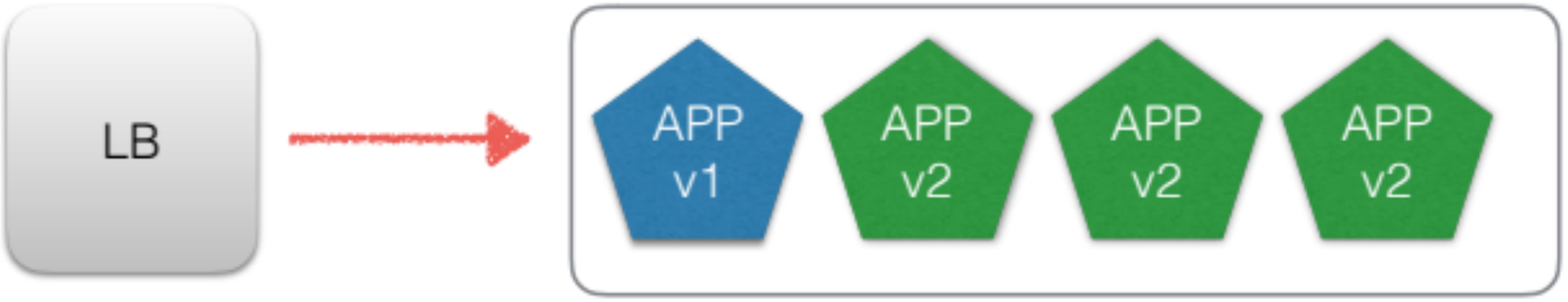
Обновить конфигурацию сервиса:

- `docker stack deploy -c docker-compose.yml …`
- `docker service update …`

Rolling Update



Rolling Update



Rolling Update

как обновлять задачи ?

```
ui:
    image: ${USER_NAME}/ui:${UI_VERSION}
    deploy:
      update_config:
        parallelism: 2
        delay: 5s
        failure_action: rollback
        monitor: 5s
        max_failure_ratio: 2
        order: start-first
```

parallelism

Сколько контейнеров обновить одновременно?

```
ui:
   image: ${USER_NAME}/ui:${UI_VERSION}
   deploy:
     update_config:
       parallelism: 2
```

Delay

Сколько ждать между обновлениями групп контейнеров?

```
ui:
   image: ${USER_NAME}/ui:${UI_VERSION}
   deploy:
     update_config:
       parallelism: 2
       delay: 5s
```

failure_action

Что делать, если обновление прошло с ошибкой?

- rollback - откатить все задачи на предыдущую версию
- pause (default) - приостановить обновление
- continue - продолжить обновление

```
ui:
   image: ${USER_NAME}/ui:${UI_VERSION}
   deploy:
     update_config:
       parallelism: 2
       delay: 5s
       failure_action: rollback
```

order

В каком порядке обновлять?

- stop-first (default) - остановить старые задачи, перед стартом новых
- start-first - запустить новые задачи, затем остановить старые

```
ui:
   image: ${USER_NAME}/ui:${UI_VERSION}
   deploy:
     update_config:
       parallelism: 2
       delay: 5s
       failure_action: rollback
       order: start-first
```

Примечание!
Работает только в compose **3.4**

Restart policy

Что **делать**, если задача завершилась ? **Перезапустить**!

Когда ?

- any - **всегда**
- on-failure - перезапускать, если приложение вернуло **код != 0**
- none - никогда

```
ui:
    image: ${USER_NAME}/ui:${UI_VERSION}
    deploy:
      restart_policy:
        condition: on-failure
        delay: 5s
        max_attempts: 3
        window: 120s
```

Limits

сколько ресурсов выделить на задачу (CPU, Memory) ?

```
ui:
   image: ${USER_NAME}/ui:${UI_VERSION}
   deploy:
     resources:
       limits:
         cpus: '0.50'
         memory: 50M
       reservations:
         cpus: '0.25'
         memory: 20M
```

Сколько **максимум** можно на один контейнер

Сколько **минимум** нужно на один контейнер

Healthchecks

Как проверить, что задача работает правильно?

```
ui:
  healthcheck:
    test: ["CMD", "curl", "-f", "http://localhost"]
    interval: 1m30s
    timeout: 10s
    retries: 3
```

Либо **HEALTHCHECK CMD** в Dockerfile

Healthchecks

Что делать если Healthcheck не прошел?