Полное руководство по миграции на SDN SPRUT

В данном руководстве описаны причины и премущества миграции с SDN Neutron, на SDN Sprut.
Приведены лучшие практики, типовые кейсы и способы миграции сервисов VK CLOUD.

Содержание

Для чего нужна миграция

Что такое sdn

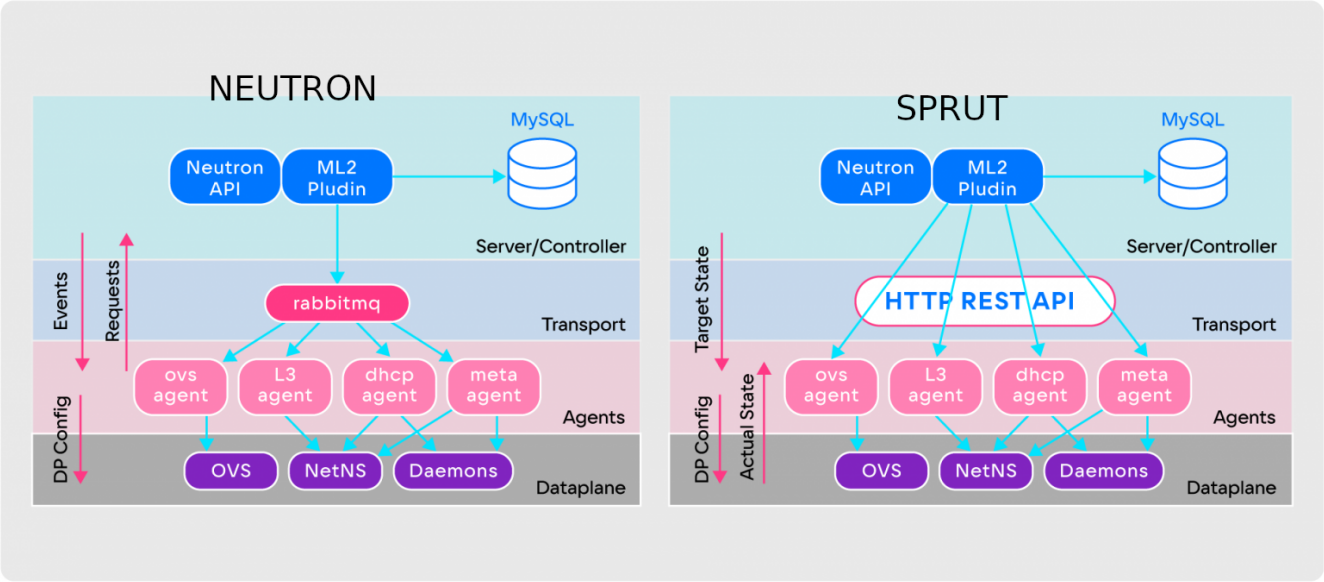
Software Defined Network — концепция выведения сетевых функций из специализированного железа на программный уровень и дальнейшего разделения ответственности на разные слои.
За счёт SDN в облаке реализуется маршрутизация, firewall и сетевая связность между сервисами в целом.

blocked URL

Чем sprut отличается от neutron

Долгое время в vk cloud использовался neutron, sdn входящий в openstack, но параллельно разрабатывалось собственное решение sdn - sprut, которое решает проблемы в архитектуре neutron, связанные с масштабируемостью и надёжностью.

На данной схеме предоставлена архитектура и схема работы обоих SDN. Подробнее в отдельной статье.



1. Для sprut агентов предусмотрена возможность постоянного сбора информации о настройках Data plane при помощи HTTP API. В neutron конфигурация доставляется при помощи очереди, без проверки что она действительно пришла. В случае ошибок в конфигурации необходима пересборка портов, то есть full sync, который занимает время.
2. В sprut больше нет событийной модели общения между компонентами. Теперь агенты всегда получают от сервера целевое состояние, в котором должны быть, и непрерывно перезапрашивают его. Получился аналог постоянного Full Sync, при котором агенты сравнивают текущее состояние Data plane с целевым состоянием от сервера, накладывают необходимый diff на Data plane и приводят его к актуальному состоянию. В теории автоматического управления такой подход называют замкнутым контуром управления.
3. RabbitMQ заменён обычным HTTP REST API. Он лучше справляется с большими массивами данных о таргетном состоянии агентов, его проще разрабатывать и мониторить. Пропала ещё одна потенциальная точка отказа.

Преимущества миграции

1. Повышение отказоустойчивости инфраструктуры, быстрое восстановление портов в случае выхода из строя сетевой инфраструктуры.
2. Переход на новый поддерживаемый sdn.

Сбор предварительных данных перед миграцией

В начале необходимо собрать информацию по сетевой инфраструктуре всех проектов при помощи опросника.

1. Используется ли в облачном проекте Shared Network?
2. Используется ли в облачном проекте Shadow Port?
3. Используется ли в облачном проекте Floating IP?
4. Используется ли в облачном проекте публичный DNS?
5. Используется ли прямое подключение к сети ext-net?
6. Использует ли клиент сервис GSLB (Global Server Load Balancer)? **ВАЖНО!** Сервис является внешним по отношении к инфраструктуре VK Cloud.
7. Используется ли в облачном проекте IPsec VPNaaS?
8. Наличие статических маршрутов на базе Subnet / Router.
9. Наличие виртуальных машин пропускающих трафик (маршрутизаторы / балансировщики / прокси / межсетевые экраны).
10. Используются ли платформенные балансировщики нагрузки?
11. Используются ли правила L7 для платформенных балансировщиков? Если да, то какие именно правила (список)?
12. Есть ли сертификаты на платформенных балансировщиках?
13. Количество кастомных групп безопасности (Security Groups) и их привязка к портам ВМ.
14. Имеется ли в облачном проекте сетевой стык (приложить ссылку в Confluence)?
15. Уровень взаимосвязей между рассматриваемым в облачным проектом и другими облачными проектами заказчика.
16. Какие платформенные сервисы используются в облачном проекте?
17. Используется ли файловое хранилище nfs/cifs?

Ограничения, к которым нужно быть готовым

Миграция с neutron на sprut процесс, требующий дополнительные действия с точки зрения сетевой инфраструктуры.

Смена плавающих ip

Описание

Для предоставления белых ip адресов (плавающих, либо получаемых при прямом подключении) в SDN NEUTRON используется сеть **ext-net**, в которой имеются подсети с диапазоном белых ip адресов.
Для SDN SPRUT используется такая же сеть, но с названием **internet** и с другим диапазоном адресов подстей.

Решение

В ходе миграции нужно учитывать замену белых ip. Белые плавающие ip можно создать заранее и передать сторонним организациям (если такие есть), чтобы они заранее ввели новые ip в белые списки

Как мигрировать

Процесс миграции может быть выполнен различными инструментами, в зависимости от сервиса и времени техокна (время когда сервис не будет работать).

Самый простой способ - пересоздание инфраструктуры в терраформе. В таком случае необходимо выполнить бэкапы/снапшоты тех вм и баз данных, где хранится состояние, так как терраформ при пересоздании удалит исходные жёсткие диски.

Другие способы, требующие меньшего техокна, будут описаны ниже.

IaaS

Сети

общая схема

В начале необходимо создать на sprut аналогичные базовые сетевые сущности, которые были на netron:

1. Маршрутизаторы со всеми статическими маршрутами
2. Сети и подсети с такими же адресами сети, шлюзами, маршрутами.
   Случай, когда в подсети подключен одновременно PaaS и IaaS рассмотрен ниже во вкладке DbaaS.
3. Секьюрити группы. Можно создать и скопировать правила при помощи скрипта

```
      #!/bin/bash

# Function to display help message
display_help() {
    echo "Usage: $0 --group-mapping=neutron_id1=sprut_id1,neutron_id2=sprut_id2,... --groups=group_name1,
group_name2,..."
    echo
    echo "Options:"
    echo "  --group-mapping    Specifies the mapping of Neutron security group IDs to Sprut security
group IDs."
    echo "  --groups           Specifies the names of security groups to be copied from Neutron to
Sprut."
    echo
    echo "Example:"
    echo "  $0 --group-mapping=e70baf6b=5a60883e-c165-4f2d-9477-4c417acd5d6f,4b345b50-df8b-4e21-8fd5-
29c90c6d4918=8492ee54-a0a6-4dd1-a8af-0c73d4b5edf5 --groups=test-neutron-sg"
    echo
    echo "This script copies security group rules from the source Neutron environment to the target
Sprut environment."
}

if ! command -v jq &> /dev/null; then
    echo "Error: jq is not installed."
    echo "jq is a lightweight and flexible command-line JSON processor."
```

```
    echo "You can install jq using the following commands:"
    echo "For Ubuntu/Debian:"
    echo "  sudo apt-get update"
    echo "  sudo apt-get install -y jq"
    echo "For CentOS/RHEL:"
    echo "  sudo yum install -y epel-release"
    echo "  sudo yum install -y jq"
    echo "For macOS using Homebrew:"
    echo "  brew install jq"
    exit 1
fi

# Function to get the authentication token
get_auth_token() {
    local token=$(openstack token issue -c id -f value)
    echo "$token"
}

# Function to check if a security group exists and get its ID
get_sg_id() {
    local sg_name="$1"
    local sg_id=$(openstack security group show "$sg_name" -f value -c id 2>/dev/null)
    echo "$sg_id"
}

# Function to create security group with '-sprut' postfix
create_sg() {
    local sg_name="$1"
    local new_sg_name="${sg_name}-sprut"
    local token="$2"

    local url="https://infra.mail.ru:9696/infra/network/v2.0/security-groups"

    local data=$(cat <<EOF
{
    "backend" : "sprut",
    "security_group": {
        "name": "$new_sg_name",
        "description": "Copy of $sg_name"
    }
}
EOF
    )

    curl -s -X POST $url \
        -H "Content-Type: application/json" \
        -H "X-Auth-Token: $token" \
        -H "X-SDN: SPRUT" \
        -d "$data"
}

# Function to remove the default egress rule
remove_default_egress_rule() {
    local sg_id="$1"

    # Get the rule ID of the default egress rule
    local rule_ids=$(openstack security group rule list "$sg_id" -f json | jq -r '.[] | select(.
Direction == "egress" and ."IP Range" == "0.0.0.0/0" and ."Port Range" == "") | .ID')

    # Remove the default egress rule
    for id in $rule_ids; do
        openstack security group rule delete "$id"
    done
}

# Function to copy security group rules from one group to another
copy_sg_rules() {
    local src_sg="$1"
    local dest_sg="$2"
    local -n mapping=$3
```

```
    # Get the rule IDs of the source security group
    local rules_json=$(openstack security group show "$src_sg" -f json | jq '.rules')

    # Iterate over each rule to fetch full details
    for rule in $(echo "${rules_json}" | jq -r '.[] | @base64'); do
        _jq() {
            echo "${rule}" | base64 --decode | jq -r "${1}"
        }

        local direction=$(_jq '.direction')
        local protocol=$(_jq '.protocol')
        local port_range_min=$(_jq '.port_range_min')
        local port_range_max=$(_jq '.port_range_max')
        local ip_range=$(_jq '.remote_ip_prefix')
        local ethertype=$(_jq '.ethertype')
        local description=$(_jq '.description')
        local remote_sg=$(_jq '.remote_group_id')

        # Check for specific group names and get corresponding IDs
        if [[ "$remote_sg" != "null" ]]; then
            local old_remote_sg="$remote_sg"
            remote_sg=$(echo ${mapping[$remote_sg]})
            echo "Replacing Neutron SecurityGroup ID '$old_remote_sg' with Sprut SecurityGroup ID
'$remote_sg'"
        fi

        # Construct command for creating rule
        local cmd="openstack security group rule create $dest_sg"
        [ "$protocol" != "null" ] && cmd+=" --protocol $protocol"
        [ "$port_range_min" != "null" ] && cmd+=" --dst-port $port_range_min:$port_range_max"
        [ "$ip_range" != "null" ] && [ "$remote_sg" == "null" ] && cmd+=" --remote-ip $ip_range"
        [ "$direction" == "egress" ] && cmd+=" --egress"
        [ "$direction" == "ingress" ] && cmd+=" --ingress"
        [ "$ethertype" != "null" ] && cmd+=" --ethertype $ethertype"
        [ "$description" != "null" ] && cmd+=" --description \"$description\""
        [ "$remote_sg" != "null" ] && cmd+=" --remote-group $remote_sg"

        # Execute the command
        echo "Executing: $cmd"
        if $cmd; then
            echo "Rule created successfully for $dest_sg"
        else
            echo "Failed to create rule for $dest_sg"
        fi
    done
}

# Get the authentication token
echo "Getting auth token"
token=$(get_auth_token)

# Initialize statistics
declare -A stats
stats[success]=0
stats[fail]=0

# Read command-line arguments
while [ $# -gt 0 ]; do
    case "$1" in
        --group-mapping=*)
            group_mapping="${1#*=}"
            ;;
        --groups=*)
            groups="${1#*=}"
            ;;
        --help)
            display_help
            exit 0
            ;;
        *)
            echo "Unknown parameter: $1"
```

```
            display_help
            exit 1
            ;;
    esac
    shift
done

# Check if group mapping is provided
if [ -z "$group_mapping" ]; then
    echo "No group-mapping provided."
fi

# Parse group mapping
declare -A sg_mapping
if [ -n "$group_mapping" ]; then
    IFS=',' read -r -a mappings <<< "$group_mapping"
    for mapping in "${mappings[@]}"; do
        IFS='=' read -r neutron_id sprut_id <<< "$mapping"
        sg_mapping["$neutron_id"]="$sprut_id"
    done
fi

# Read each security group name from the command line arguments
IFS=',' read -r -a sg_names <<< "$groups"

for sg_name in "${sg_names[@]}"; do
    # Trim whitespace
    sg_name=$(echo "$sg_name" | xargs)

    echo "-----------------------------------"
    echo "Checking if group '$sg_name' exists..."

    # Check if the security group exists
    sg_id=$(get_sg_id "$sg_name")
    if [ -z "$sg_id" ]; then
        echo "No SecurityGroup found for '$sg_name'"
        stats[fail]=$((stats[fail]+1))
        continue
    fi

    new_sg_name="${sg_name}-sprut"
    echo "Checking if group '$new_sg_name' already exists..."

    # Check if the target security group with postfix exists
    new_sg_id=$(get_sg_id "$new_sg_name")
    if [ ! -z "$new_sg_id" ]; then
        echo "SecurityGroup '$new_sg_name' already exists"
        stats[fail]=$((stats[fail]+1))
        continue
    fi

    echo "Creating new security group '$new_sg_name'..."
    create_sg "$sg_name" "$token"

    echo "Removing default egress rule from '$new_sg_name'..."
    remove_default_egress_rule "$new_sg_name"

    echo "Copying rules from '$sg_name' to '$new_sg_name'..."
    copy_sg_rules "$sg_id" "$new_sg_name" sg_mapping

    echo "Copying finished for '$sg_name'"
    stats[success]=$((stats[success]+1))
done

echo "-----------------------------------"
echo "Processing complete."
echo "Statistics:"
echo "Successfully copied: ${stats[success]}"
echo "Failed to copy: ${stats[fail]}"
```

```
      ./copy-security-group.sh --group-mapping=<id    1>=<id    1>,<id    2>=<id    2>,... \
--groups=<      1>,<      2>,...
```

--group-mapping - параметр нужен для правил, где в качестве источников указан не cidr, а другая секьюрити группа. Так как исходные id групп отличаются от целевых, например группы нейтрон all и спрут all, при этом названия у них одинаковые, необходимо через личный кабинет достать id и передать их в скрипт.
--groups - список групп на нейтроне, которые будут скопированы

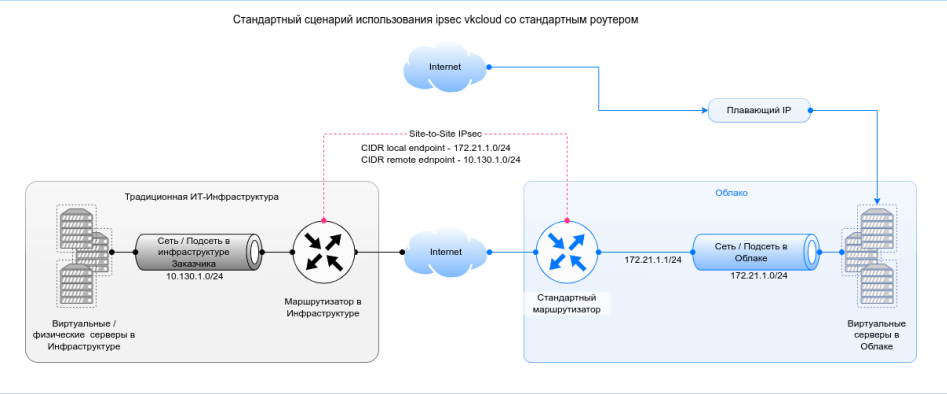
> В проекте по умолчанию доступна только группа **default-sprut** (в веб интерфейсы называется default) на sprut. Остальные группы будут доступны после создания ВМ с такими группами в сети sprut. Вм можно будет потом сразу удалить.

terraform

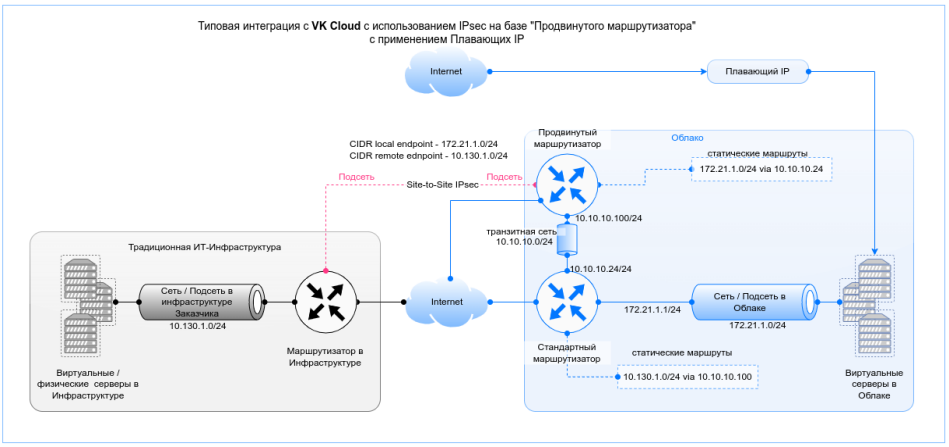
Можно скопировать описание всех сетевых объектов, которые уже имеются на neutron, добавить в названии и аттрибуте name постфикс -sprut и применить изменения, не забыв в зависимостях указать новые названия.
Сеть для интернета также будет изменена с ext-net на internet.

Ipsec

- Туннели с одинаковыми селекторами (исходные и целевые диапазоны подсетей) не могут существовать одновременно, даже если они на разных sdn. Это означает, что создать аналогичный neutron туннель на sprut заранее нельзя, так как это приведёт к проблемам в работе с исходного.
- Для миграции можно заранее заготовить манифест terraform, который будет описывать туннель на sprut, также это можно сделать для туннеля на neutron для отката миграции при необходимости. Когда будет выполнятся миграция, исходный туннель на neutron необходимо удалить и применить манифест для туннеля на sprut, либо создать его через графический интерфейс, но это займёт больше времени.
- Необходимо помнить, что туннель создаётся с обоих сторон и клиент на другой стороне должен также быть готов удалить старый и принять новый туннель.
- Для построения ipsec туннелей на sprut необходимо использовать продвинутый маршрутизатор, а не стандартный как для neutron.

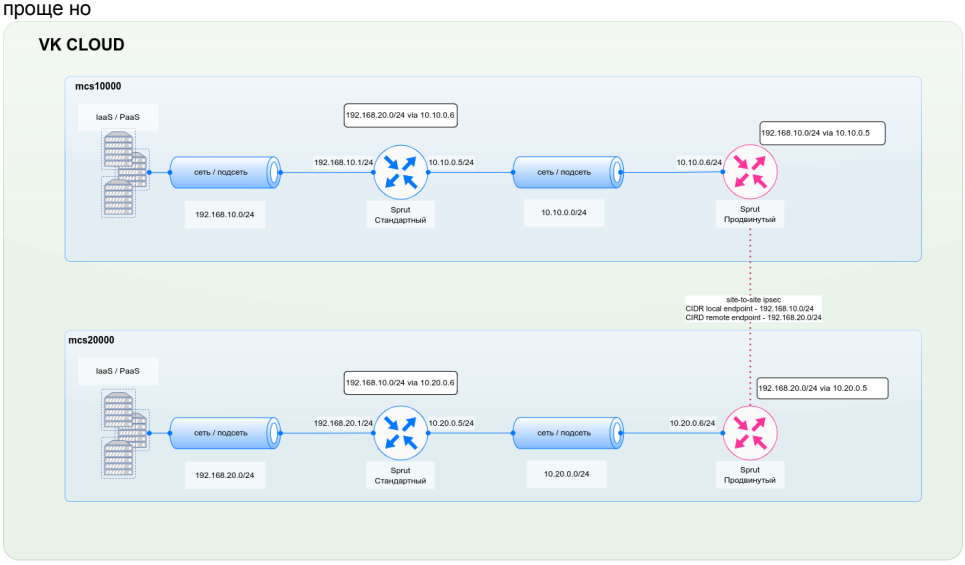


Данную схему можно преобразовать в такую, для сохранение описанного функционала



Данная схема необходима, так как сети подключенные к продвинутому роутеру не поддерживают плавающие ip адреса. Подключение продвинутого и стандартного роутера через транзитную сеть позволяет передать всю маршрутизацию роутерам. Никакие маршруты по dhcp передавать не нужно.

- В случае, если Ipsec используются для организации сетевой связности между разными проектами vk cloud переключение выполнить проще но



Рекомендуется заменить её на схему с шаренной сетью, shared network, так как это более производительно и надёжно. Стандартную сеть можно сделать доступной сразу в нескольких проектах, для этого нужно сделать запрос в техподдержку.

- Для работы SNAT, то есть наличия доступа выхода в интернет, подсеть должна быть подключена к стандартному, а не продвинутому маршрутизатору.
- К одной сети нельзя подключить несколько стандартных роутеров без админских прав, которые есть у техподдержки.
- Мы используем продвинутые роутеры для подключения проектов к шаренной сети.
- Между стандартным и продвинутым маршрутизатором мы строим транзитную сеть. Для каждого стандартного маршрутизатора нужна отдельная транзитная сеть.
- Необходимо прописать соответствующие статические маршруты на стандартных и продвинутых маршрутизаторах.

Балансировщики

- Сервис одинаков для neutron и sprut, но для миграции его необходимо пересоздать, так как у балансировщика нельзя заменить сеть.
- Можно пересоздать в terraform, указав сеть sprut в описании манифеста и подключить виртуальные машины уже после их миграции.
- Можно создать балансировщик вручную и указать вм в правилах.
- В разработке скрипт для создания копии балансировщика и с назначением всех виртуальных машин.

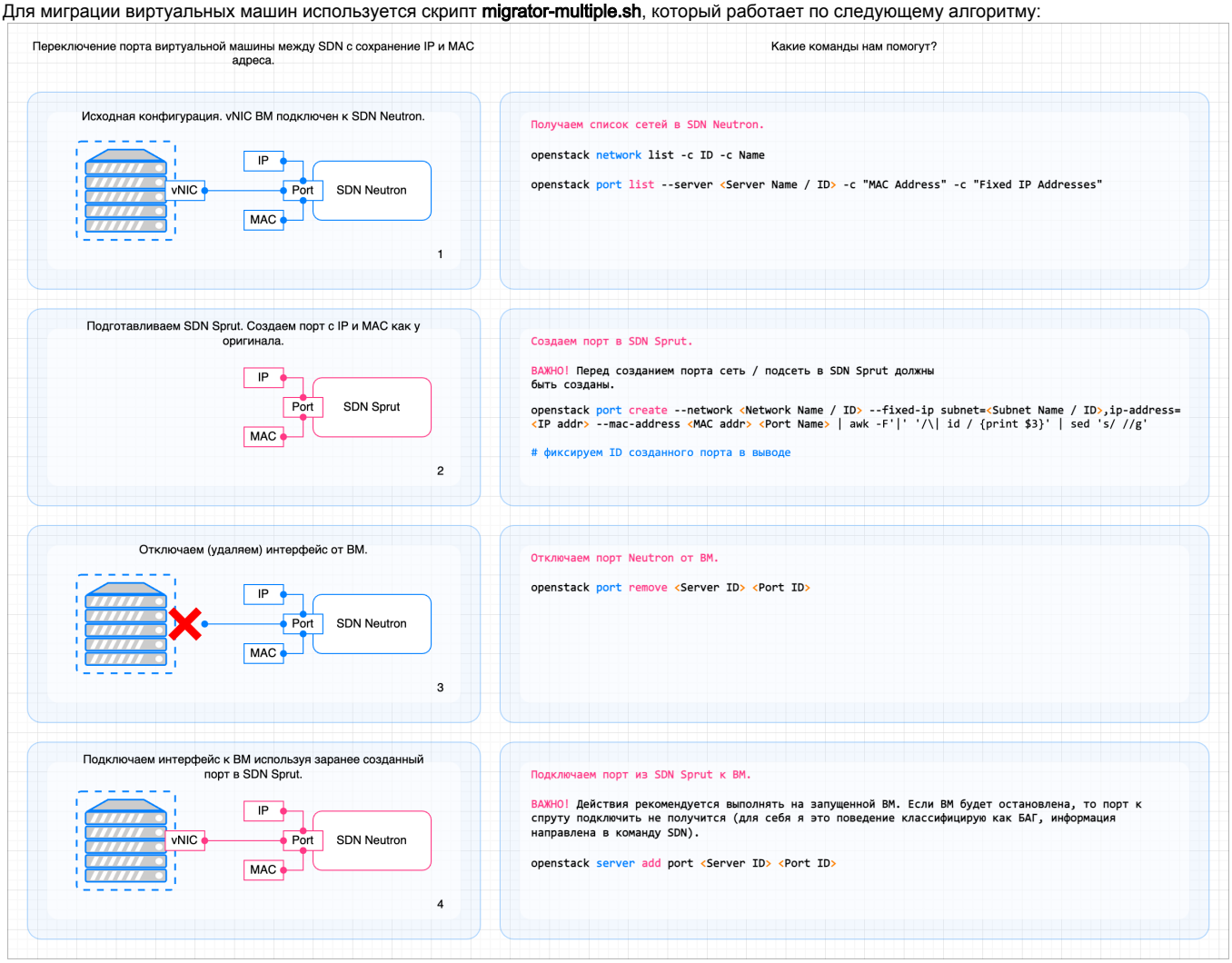
Публичный DNS

- В связи со сменой плавающих ip необходимо отредактировать текущие A-записи.
- Если используется внешний сервис dns, изменить записи в нём.

Виртуальные машины

общая схема

Для миграции виртуальных машин используется скрипт **migrator-multiple.sh**, который работает по следующему алгоритму:



Сценарий предполагает выполнение последовательности действий, которая позволит переключить сетевой интерфейс ВМ в новый SDN. При этом важно помнить, что данная операция выполняется с разрывом сетевой связности.

Классификация виртуальных машин по сложности мигрирования.

Красный - несколько сетевых интерфейсов, вм выступает в роли роутера/прокси/пограничного файерволла.
Рекомендуется вручную мигрировать данную виртуальную машину без скрипта, так как скрипт предназначен для вм с одним портом.

Жёлтый - вм с плавающим ip либо с прямым подключением к сети ext-net.
Необходимо создать заранее плавающий ip в sprut и указать его id в конфиг файле скрипта.
В случае с прямым подключением ext-net рекомендуется перейти на плавающий ip, либо пересоздать вм.

Зелёный - один сетевой интерфейс в серой сети без плавающего ip.

Подготовительные шаги.

1. В проекте VK Cloud создана ВМ с подключением к виртуальной сети, принадлежащей SDN Neutron.
2. В проекте VK Cloud создана новая сеть в SDN Sprut с подсетью, настройки которой повторяют настройки подсети, созданной в сети Neutron.
3. Подготовлено рабочее место администратора (ВМ с ОС Linux) с установленными компонентами OpenStack CLI, файл конфигурации отправлен в source и вызовы CLI проходят без ошибок.
4. На рабочее место администратора перенесен скрипт автоматизации.
5. Определен сервер, порт которого будет переключаться (логическое имя сервера).
6. Выполнена проверка о наличии аналогичных нейтроновским секьюрити групп на спруте. Их имя должно иметь постфикс -sprut (исключение базовые группы создаваемые по умолчанию вроде: default,all).
   Скрипт:

```
      #!/bin/bash

echo "
########################################
#                                      #
#      Security Group Check Script     #
#                                      #
########################################

This script checks all VMs in the tenant, collects the names of the assigned security groups, and
verifies if there are corresponding security groups with the '-sprut' postfix.

It will skip checking for security groups named 'default', 'ssh+www', and 'all'.

"

# Function to get a list of all VMs in the tenant
function get_all_vms {
    openstack server list -f value -c Name
}

# Function to get the security groups assigned to a VM
function get_vm_security_groups {
    local vm_name=$1
    openstack server show "$vm_name" -f json | jq -r '.security_groups[] | .name'
}

# Function to check if a security group with a given name and '-sprut' postfix exists
function check_sprut_sg_exists {
    local sg_name=$1
    openstack security group list -f value -c Name | grep -qw "${sg_name}-sprut"
}

# Main script execution
all_vms=$(get_all_vms)
sg_names=()

for vm in $all_vms; do
    echo "Checking VM: $vm"
    vm_sg_names=$(get_vm_security_groups "$vm")
    echo "Security groups found on VM $vm: $vm_sg_names"
    for sg_name in $vm_sg_names; do
        sg_names+=("$sg_name")
```

```
    done
done

# Remove duplicates and sort the list
unique_sg_names=($(echo "${sg_names[@]}" | tr ' ' '\n' | sort -u | tr '\n' ' '))

# Filter out the security groups that should not be checked
filtered_sg_names=()
for sg_name in "${unique_sg_names[@]}"; do
    if [[ "$sg_name" != "default" && "$sg_name" != "ssh+www" && "$sg_name" != "all" ]]; then
        filtered_sg_names+=("$sg_name")
    fi
done

# Check for corresponding '-sprut' security groups and report missing ones
missing_sg=()
for sg_name in "${filtered_sg_names[@]}"; do
    echo "Checking for corresponding '-sprut' security group for: $sg_name"
    if check_sprut_sg_exists "$sg_name"; then
        echo "Found corresponding '-sprut' group for: $sg_name"
    else
        echo "Missing corresponding '-sprut' group for: $sg_name"
        missing_sg+=("$sg_name")
    fi
done

echo "-----------------------------------"
echo "Security Group Check Summary"
echo "-----------------------------------"
if [ ${#missing_sg[@]} -eq 0 ]; then
    echo "All security groups have corresponding '-sprut' groups."
else
    echo "The following security groups do not have corresponding '-sprut' groups:"
    for sg in "${missing_sg[@]}"; do
        echo "- $sg"
    done
fi
echo "-----------------------------------"
```

7. Запустить скрипт миграции:

Для запуска скрипта:

```
./migrator-multiple.sh --all-secgroup-sprut-id=<id  all  >  <csv     >
```

--all-secgroup-sprut-id - так как в проекте будет 2 группы all для neutron и sprut, необходимо указать id в sprut, так как openstack cli неспособен различить принадлежность секьюрити группы к sdn.

csv файл с описанием мигрируемых вм имеет следующий формат:

имя вм1,имя сети sprut,имя подсети sprut,<опционально: id плавающего ip на спрут, который назначится>
имя вм2,имя сети sprut,имя подсети sprut

Код скрипта:

```
#!/bin/bash

echo "
########################################
#                                      #
#        Port Migration Script         #
#                                      #
```

```
########################################

Mandatory Input Data:

Input file format:
server_name1,dest_net1,dest_subnet1,floating_ip_id1
server_name2,dest_net2,dest_subnet2,floating_ip_id2

Note: If floating_ip_id is not provided, the script will not attach a Floating IP.

Optional:
--all-secgroup-sprut-id=<id>
--ssh-www-secgroup-sprut-id=<id>

"

# Parse arguments
for i in "$@"
do
case $i in
    --all-secgroup-sprut-id=*)
    all_sg_sprut_id="${i#*=}"
    shift
    ;;
    --ssh-www-secgroup-sprut-id=*)
    ssh_www_sg_sprut_id="${i#*=}"
    shift
    ;;
    *)
    # unknown option
    ;;
esac
done

# Function Definitions

# Capture port information with full details
function capture_info_full {
    echo "Executing step 1: Capturing port information"

    # Define the migrated port name format
    migrated_port_name="${sname}_migrated_port"

    # Check if the migrated port already exists and is attached to the server
    existing_migrated_port_info=$(openstack port list -f value -c ID -c Name | grep "$migrated_port_name")
    existing_migrated_port_id=$(echo "$existing_migrated_port_info" | awk '{print $1}' | head -n 1)

    if [ ! -z "$existing_migrated_port_id" ]; then
        # Check if this port is already attached to the server
        echo "Migrated port ${migrated_port_name} exists, checking for attachment..."
        attached_port_info=$(openstack server port list $sname -f value -c ID | grep
"$existing_migrated_port_id")
        if [ ! -z "$attached_port_info" ]; then
            echo "Migrated port $migrated_port_name already exists and is attached to server $sname.
Skipping..."
            echo "*****************************************"
            return 1 # Use return code 1 to indicate skipping
        fi
    fi

    # Proceed with capturing port information if no migrated port is attached
    port_output=$(openstack port list --server $sname -c id -c "MAC Address" -c "Fixed IP Addresses")
    srcpid=$(echo "$port_output" | awk -F'|' 'NR==4{print $2}' | sed 's/ //g')
    mcs=$(echo "$port_output" | awk -F'|' 'NR==4{print $3}' | sed 's/ //g')
    ips=$(echo "$port_output" | awk -F'|' 'NR==4{print $4}' | grep -oP "ip_address='\K[^']+")

    echo "Source Port ID is:        $srcpid"
    echo "Source Port IP Addr is:   $ips"
    echo "Source Port MAC Addr is:  $mcs"
    echo "*****************************************"
}
```

```
# Capture server ID and security group names
function capture_id_and_sec_group {
    echo "Executing step 2: Capturing server ID and security group names"

    server_output=$(openstack server show $sname)
    servid=$(echo "$server_output" | awk -F'|' '/\| id / {print $3}' | sed 's/ //g')
    echo "Server ID is:             $servid"

    # Fetch security group IDs
    sec_group_ids=$(openstack port show $srcpid -c security_group_ids -f value)

    # Preprocess to remove brackets and split by comma
    sec_group_ids=$(echo $sec_group_ids | tr -d '[]' | tr -d '"' | tr -d "'")

    # Convert to array and iterate
    IFS=',' read -ra ADDR <<< "$sec_group_ids"
    sec_group_names=()
    for sec_group_id in "${ADDR[@]}"; do
        sec_group_name=$(openstack security group show $sec_group_id -c name -f value)
        sec_group_names+=("$sec_group_name")
    done
    echo "Security Groups captured: ${sec_group_names[@]}"
    echo "*****************************************"
}

# Create port in target network with source IP and MAC, with checks for existing port by name using grep
function create_port_with_mac_ip {
    echo "Executing step 3: Creating port with source IP and MAC"

    # Define the port name format
    port_name="${sname}_migrated_port"

    # Attempt to find an existing port by name using grep
    existing_port_info=$(openstack port list -f value -c ID -c Name | grep "$port_name")
    existing_port_id=$(echo "$existing_port_info" | awk '{print $1}' | head -n 1)

    if [ ! -z "$existing_port_id" ]; then
        echo "Port named $port_name already exists. Port ID: $existing_port_id"
        pmigid=$existing_port_id
    else
        # If no existing port found, attempt to create a new port
        create_port_cmd="openstack port create --network $defnet --fixed-ip subnet=$defsubnet,ip-address=$ips --
mac-address=$mcs $port_name"
        echo "Running command: $create_port_cmd"
        new_port_output=$($create_port_cmd)
        pmigid=$(echo "$new_port_output" | awk -F'|' '/\| id / {print $3}' | sed 's/ //g')
        echo "Step 3 complete (New Port Created)"
    fi
    echo "*****************************************"
}

# Disconnect existing port from server
function detach_source_port {
    echo "Executing step 4: Disconnecting existing port from server"

    detach_port_cmd="openstack server remove port $servid $srcpid"
    echo "Running command: $detach_port_cmd"
    $detach_port_cmd
    echo "Step 4 complete (Source port disconnected from server $sname)"
    echo "*****************************************"
}

# Connect new port to server
function attach_new_port {
    echo "Executing step 5: Connecting new port to server"

    attach_port_cmd="openstack server add port $servid $pmigid"
    echo "Running command: $attach_port_cmd"
    $attach_port_cmd
    echo "Step 5 complete (New port attached to server)"
```

```
    echo "*****************************************"
}

# Set security groups on the new port
function set_security_groups {
    echo "Executing step 6: Setting security groups on new port"
    echo "Setting captured groups: ${sec_group_names[@]}"

    for sec_group_name in "${sec_group_names[@]}"; do
        echo "Original Security Group: $sec_group_name"
        if [[ "$sec_group_name" == "all" ]]; then
            modified_sec_group_id="$all_sg_sprut_id"
        elif [[ "$sec_group_name" == "default" ]]; then
            modified_sec_group_name="default-sprut"
            if openstack security group show "$modified_sec_group_name" -c id -f value &> /dev/null; then
                modified_sec_group_id=$(openstack security group show "$modified_sec_group_name" -c id -f value)
            else
                echo "Security Group $modified_sec_group_name does not exist, skipping..."
                continue
            fi
        elif [[ "$sec_group_name" == "ssh+www" ]]; then
            modified_sec_group_id="$ssh_www_sg_sprut_id"
        else
            modified_sec_group_name="${sec_group_name}-sprut"
            # Check if modified security group exists before setting it
            if openstack security group show "$modified_sec_group_name" -c id -f value &> /dev/null; then
                modified_sec_group_id=$(openstack security group show "$modified_sec_group_name" -c id -f value)
            else
                echo "Security Group $modified_sec_group_name does not exist, skipping..."
                continue
            fi
        fi
        echo "Modified Security Group ID: $modified_sec_group_id"
        set_sg_cmd="openstack port set --security-group $modified_sec_group_id $pmigid"
        echo "Running command: $set_sg_cmd"
        $set_sg_cmd
        echo "Security Group $sec_group_name set on new port"
    done
    echo "Step 6 complete (Security groups assignment complete)"
    echo "*****************************************"
}

# Attach Floating IP to the new port if provided
function attach_floating_ip {
    if [ ! -z "$floating_ip_id" ]; then
        echo "Executing step 7: Attaching Floating IP"

        attach_fip_cmd="openstack floating ip set --port $pmigid $floating_ip_id"
        echo "Running command: $attach_fip_cmd"
        $attach_fip_cmd
        echo "Floating IP $floating_ip_id attached to new port"
        echo "*****************************************"
    fi
}

# Process migration for each server
function process_migration {
    sname=$1
    defnet=$2
    defsubnet=$3
    floating_ip_id=$4

    echo "Processing migration for server: $sname"
    capture_info_full
    if [ $? -eq 1 ]; then
        # Skipping logic, e.g., continue in a loop
        continue
    fi
    capture_id_and_sec_group
    create_port_with_mac_ip
    detach_source_port
```

```
    attach_new_port
    set_security_groups
    attach_floating_ip
    echo "Migration completed for server: $sname"
    echo "----------------------------------------"
}

# Main script execution

if [ -z "$1" ]; then
    echo "Error: No input file provided."
    exit 1
fi

start_time=$(date +%s)

while IFS=, read -r server_name dest_net dest_subnet floating_ip_id
do
    process_migration "$server_name" "$dest_net" "$dest_subnet" "$floating_ip_id"
done < "$1"

end_time=$(date +%s)
elapsed_time=$(($end_time - $start_time))
echo "Elapsed time: $elapsed_time seconds"
```

После миграции может возникнуть необходимость перезагрузить dhclient на вм

Для Windows:

```
ipconfig /release
ipconfig /renew
```

Для Lunix:
```bash
dhclient
```

terraform

Как было сказано ранее, можно мигрировать вм на другой sdn при помощи terraform, что требует пересоздания вм. Миграция через скрипт, описанный выше также приведёт к пересозданию вм, если выполнить **terraform apply**.
Это возникает из-за того, что id портов вм меняются, так как скрипт создаёт новые порты. Чтобы избежать пересоздания, необходимо поправить state указав новую сеть, на которую ссылается terraform, а также исправить state файл.
Вм может быть много и менять стейт вручную неудобно, следующий скрипт позволяет собрать информацию о вм, которые были мигрированы и подготовить новый скрипт, правящий состояние terraform:

```
    #!/bin/bash

echo "
########################################
#                                      #
#  Terraform State Modification Script#
#                                      #
########################################
"

# ,
if [[ -n "$1" ]]; then
    state_file="-state=$1"
    echo "Using specified state file: $1"
else
    state_file=""
    echo "Using default state file"
```

```
fi

#       Terraform
terraform_state_list=$(terraform state list $state_file)

#  ,    vkcs_compute_instance
compute_instances=$(echo "$terraform_state_list" | grep "vkcs_compute_instance")

#   compute_instances
if [[ -z "$compute_instances" ]]; then
    echo "No vkcs_compute_instance resources found in the current Terraform state."
    exit 1
fi

#
output_script="terraform_modify_to_sprut_state.sh"
echo "#!/bin/bash" > $output_script
echo "echo '########################################'" >> $output_script
echo "echo '#                                      #'" >> $output_script
echo "echo '#  Modifying Terraform State Script   #'" >> $output_script
echo "echo '#                                      #'" >> $output_script
echo "echo '########################################'" >> $output_script

#
table_output="terraform resource name | openstack vm id | vm name\n"

#    compute_instance
while IFS= read -r instance; do
    #  id
    resource_info=$(terraform state show $state_file $instance)
    resource_id=$(echo "$resource_info" | grep -E '^\s*id\s*=' | awk -F' = ' '{print $2}' | tr -d '"' | head -n
1)
    resource_name=$(echo "$resource_info" | grep -E '^\s*name\s*=' | awk -F' = ' '{print $2}' | tr -d '"' |
head -n 1)

    #   id
    if [[ -z "$resource_id" || -z "$resource_name" ]]; then
        echo "No id or name found for $instance in the current Terraform state."
        continue
    fi

    #
    echo "echo 'Modifying $instance'" >> $output_script
    echo "terraform state rm $instance $state_file" >> $output_script
    echo "terraform import $instance $resource_id $state_file" >> $output_script

    #
    table_output+="$instance | $resource_id | $resource_name\n"
done <<< "$compute_instances"

#
chmod +x $output_script

#
echo -e "Generated script: $output_script\n"
echo -e "$table_output"

#
table_output_file="state_modification_table.txt"
echo -e "$table_output" > $table_output_file

echo "Table of resources saved to $table_output_file"
```

Для запуска:

```
modify_terraform_state.sh <  >
```

Если не указать в параметрах стейт файл терраформа, будет использоваться текущий файл **terraform.tfstate**

Необходимо в коде терраформе также указать новые сети у вм.

NFS/CIFS

общая схема

Файловое хранилище поддерживает бэкапы, однако их нельзя поднять в сети другого sdn.
Необходимо:

1. Создать аналогичное хранилище в sprut сети.
2. Создать вм 1-2 и подключить её к сетям старого и нового хранилищь.
3. Выполнить подключение на вм к обоим хранилищам и запустить rsync и перелить данные в новое хранилище.

PaaS

Общая схема миграции подразумевает создание копии PaaS в сети neutron и переноса нагрузки при помощи встроенных средств бэкапирования.
Для простоты решения рекомендуется остановить кластеры на запись, пока выполняется перенос.

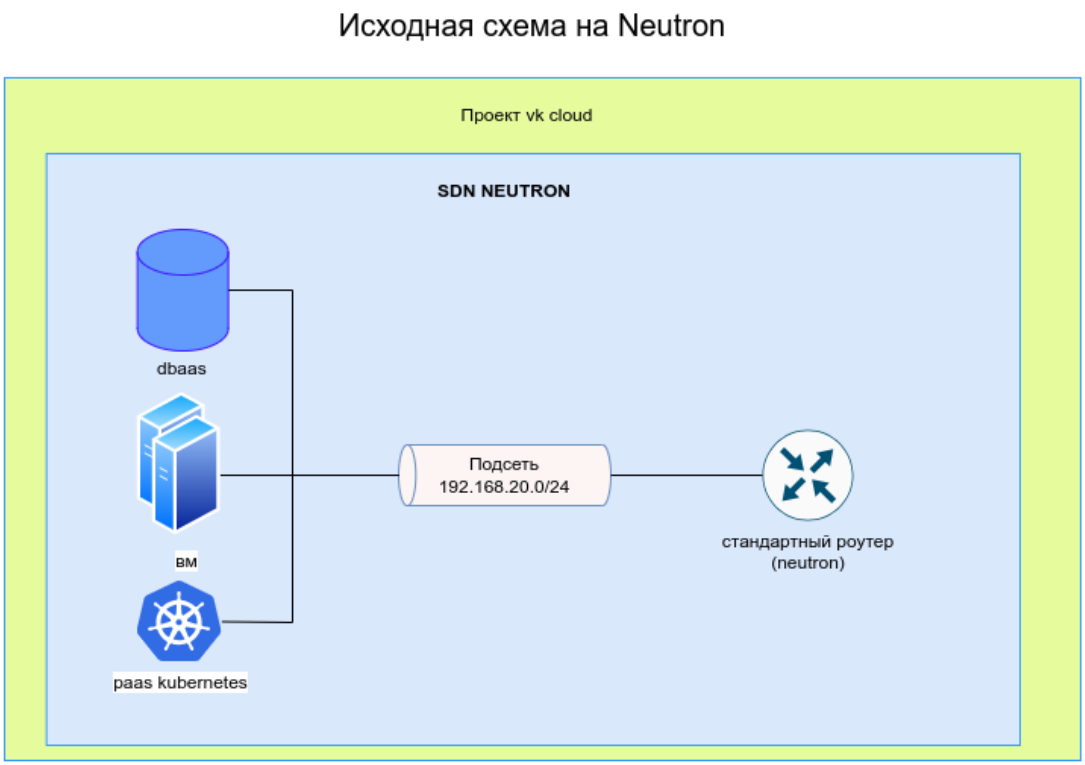
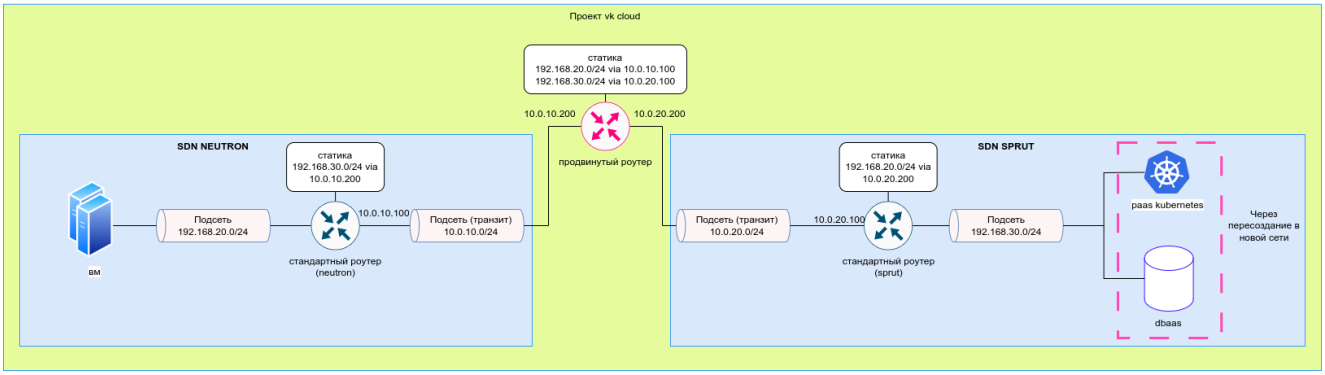


Схема PaaS на Sprut, IaaS на neutron



В данной схеме можно использовать продвинутый роутер в качестве связности neutron и sprut.

Kubernetes

общая схема

1. Создаём аналогичный кластер в новой сети sprut. В случае если в сети помимо кластера kubernetes есть другие сервисы или виртуалки, адрес сети на sprut, где размещён kubernetes должен отличаться от исходного.
2. Переносим нагрузку при помощи средства бэкапирования velero, в том числе pv. Примеры использования velero находятся в инструкции: https://github.com/IlyaNyrkov/k8s-velero-vkcloud-workshop
3. Внешние ip балансировщиков будут другие на новом кластере.
4. Подключаем продвинутый роутер к транзитным сетям с исходным и новым роутером на спруте. Прописываем необходимую статику как на схеме выше.
5. Проверяем функционирование приложения.
6. Удаляем старый кластер.

DbaaS

1. Останавливаем исходную базу данных на запись и делаем снапшот. Альтернативно можно использовать pgdump и другие встроенные средства бэкапирования баз данных.
2. Из бэкапа поднимаем базу данных в новой сети.
3. Строем перемычку с виртуальными машиными при помощи продвинутого роутера.
4. Правим конфиги вм, чтобы они ходили в бд с новым адресом.
5. Удаляем исходную бд.