

Архитектура вычислительных систем

Оркестрация, docker compose

Романюта Алексей Андреевич

alexey-r.98@yandex.ru

Кафедра вычислительных систем
Сибирский государственный университет телекоммуникаций и информатики



Оркестрация

- Оркестратор – ПО, занимающееся запуском, размещением и отслеживанием работы экземпляров приложений
- Systemd, supervisord, docker/docker compose, kubernetes, proxmox, cloud platforms

Systemd

- В linux используется система инициализации systemd (Ранее был init.d)
- Systemd умеет:
 - Отслеживать состояние процесса
 - Перезапускать при различных условиях
 - Управлять cgroup для процесса
 - Работать с домашними каталогами (systemd-homed)
 - Управление конфигурацией dns (systemd-resolved)
 - Таймеры (systemd-timers) – замена cron с более гибким условием

Supervisord

- В отличие от systemd направлена на управление конкретно процессами
- Может поддерживать процессы реализующие воркеры
- Иногда используется как сложная система управления в контейнерах – подсистема контейнеров следит за работой supervisord, supervisord следит за остальным. Пример – nginx в связке с php-fpm

Docker compose

- Docker-compose – инструмент для управления множеством контейнеров
- Запускаемые сервисы описываются с помощью yaml-формата

```
version: '2.1' # это только для v1 (pip)
services:
  nginx:
    image: nginx:1.26.0
    container_name: "moira-nginx"
    restart: unless-stopped
    links:
      - redis:redis-container
    ports:
      - "8088:8080"
    depends_on:
      - redis
    volumes:
      - ./nginx.conf:/etc/nginx/conf.d/default.conf
```

```
redis:
  image: redis:6.2.6-alpine
  restart: unless-stopped
```

- Version 3 – ориентирована на Docker Swarm
- Version 2 – для запуска на хост
- Deprecated field

- Поле version применимо только к docker-compose 1 (pip пакет)

Docker compose

- `docker-compose` – Реализация v1 в качестве pip пакета. Совместимо с `ansible` модулем `community.docker.docker_compose`, но - deprecated
- `docker compose` – Реализация v2 в качестве плагина и расширения `docker`. Совместимо с v1, но при чередовании вызывают пересоздание контейнеров. Совместимо с `ansible` модулем `community.docker.docker_compose_v2`
- Docker поддерживает кластеризацию – `docker swarm`. Хосту назначается роль – `manager` или `worker`
- Развертывание «стека» в `swarm` производится через `compose`

Docker compose: основные команды

- `docker compose up ${service}` – запустить сервис `${service}`. Если сервис не указан, запускаются все сервисы. Ключ `-d` возвращает управление в консоль
- `docker compose down` – остановить и удалить все контейнеры, описанные в конфигурации
- `docker compose stop ${service}` – остановить сервис `${service}`. Если сервис не указан, останавливаются все сервисы
- `docker compose logs ${service}` – вывести лог сервиса `${service}`. Если сервис не указан, выводится лог всех сервисов. Использование ключа `-f` позволяет выводить лог в реальном времени
- `docker compose ps` – вывод статуса контейнеров, описанных в конфигурации

Docker compose: основные команды

- `docker compose config` – команда позволяет вывести итоговую конфигурацию `docker compose`, который будет применен. Эта команда раскрывает все подстановки и объединяет все файлы `docker compose`, если их указано больше одного.

Docker-compose: синтаксис

- Каждый файл содержит следующие основные поля - `services`
- Поле `services` содержит список сервисов для данного `compose` файла

```
services:
  nginx:
    image: nginx:1.20.2
    container_name: "moira-nginx"
    restart: unless-stopped
    links:
      - redis:redis-container
    ports:
      - "8088:8080"
    depends_on:
      - redis
    volumes:
      -
        ./configs/nginx.conf:/etc/nginx/conf.d/default.conf
```

Docker-compose: синтаксис

```
services:                                # Начало секции списка сервисов
  nginx:                                 # Сервис nginx
    image: nginx:1.20.2                  # Образ, из которого контейнер будет запущен
    container_name: "moira-nginx"        # Имя контейнера – по умолчанию автоматически
    restart: unless-stopped              # Политика рестарта
    links:                                # links позволяет обратиться к другим контейнерам
                                           # не по имени их сервиса
      - redis:redis-container            # Например, redis доступен как redis-container
    ports:                                # Список открытых портов. Не работает при
                                           # network_mode: host
      - "8088:8080"                      # 8088 хоста сопоставляется порту 8080 контейнера
    depends_on:                           # Зависимости по запуску – запуск только после
      - redis                             # контейнера redis
    volumes:                              # Список внешних файлов и директорий для монтирования
                                           # внутрь контейнера
      - ./configs/nginx.conf:/etc/nginx/conf.d/default.conf
redis:
  image: redis:6.2.6-alpine #
  restart: unless-stopped #
```

Docker-compose: на примерах

- Задача: Есть некий telegram-бот, у которого имеется Dockerfile. Нужно написать файл конфигурации docker-compose.yml, предназначенный для сборки и запуска экземпляра бота на локальной машине. Для работы боту необходима база данных *mongodb* и админ-панель *mongo-express* для управления БД на локальной машине
- База данных имеет скрипт инициализации `mongo-init.js`

```
services:  
  bot: ...  
  mongo-express: ...  
  mongodb: ...
```

Docker-compose: База данных

- База данных - mongodb
- База данных имеет скрипт инициализации `mongo-init.js`
- Выбор образа – mongo версии 4.2.16

```
mongodb:  
  image: mongo:4.2.16
```

```
docker-compose.yml:  
services:  
  mongodb:  
    ...
```

Docker-compose: База данных

- Определим хостнейм внутри контейнера, параметры рестарта и имя контейнера в системе

```
mongodb:
  image: mongo:4.2.16
  hostname: "r9odt-42"
  restart: unless-stopped
  container_name: "mongo"
```

```
docker-compose.yml:
services:
  mongodb:
    ...
```

Существующие политики рестарта контейнеров:

- no – Не перезапускать контейнер. Политика по умолчанию
- on-failure[:max-retries] – перезапуск контейнера при ошибках (exit-code != 0).
max-retries – максимальное количество попыток перезапустить контейнер
- always – перезапускать всегда
- unless-stopped – перезапуск всегда, кроме случая остановки командой docker stop. Так же не перезапускать при рестарте docker-daemon, если контейнер был остановлен

Docker-compose: База данных

- При первом запуске происходит инициализация базы данных – создание учетной записи администратора и первой бд. Определим пользователя и имя базы

```
mongodb:
  image: mongo:4.2.16
  hostname: "r9odt-42"
  restart: unless-stopped
  container_name: "mongo"
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: example
    MONGO_INITDB_DATABASE: tgbot
```

```
docker-compose.yml:
services:
  mongodb:
    ...
```

Docker-compose: База данных

- Определим порт, по которому будет доступна БД. На хост-системе будет доступна по адресу `127.0.0.1:27017`

```
mongodb:
  image: mongo:4.2.16
  hostname: "r9odt-42"
  restart: unless-stopped
  container_name: "mongo"
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: example
    MONGO_INITDB_DATABASE: tgbot
  ports:
    - "127.0.0.1:27017:27017/tcp"
```

```
docker-compose.yml:
services:
  mongodb:
    ...
```

Docker-compose: База данных

- В случае с mongodb можно разместить скрипт инициализации в `/docker-entrypoint-initdb.d/`

```
mongodb:
  image: mongo:4.2.16
  hostname: "r9odt-42"
  restart: unless-stopped
  container_name: "mongo"
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: example
    MONGO_INITDB_DATABASE: tgbot
  ports:
    - "127.0.0.1:27017:27017/tcp"
  volumes:
    - ./mongo-init.js:/docker-entrypoint-initdb.d/mongo-init.js:ro
```

```
mongo-init.js:
db.createUser(
  {
    user: "tgbot",
    pwd: "tgbot",
    roles: [
      {
        role: "readWrite",
        db: "tgbot"
      }
    ]
  }
);
```


Docker-compose: База данных

- Используя `docker compose up -d` запускаем базу данных

```
romanutaaa@r9odt-cat:tg-bot [05:46 ] [master] $ docker-compose up -d
Creating network "tg-bot_default" with the default driver
Creating mongo ... done
romanutaaa@r9odt-cat:tg-bot [05:46 ] [master] $ docker-compose ps
Name                  Command                  State                  Ports
-----
mongo                docker-entrypoint.sh mongod    Up                    127.0.0.1:27017->27017/tcp
romanutaaa@r9odt-cat:tg-bot [05:46 ] [master] $
```

Первый запуск создает виртуальную сеть для данного проекта – в неё входят все контейнеры, описанные в файле `docker-compose.yml` у которых явно не указана сеть

- Здесь и далее на скриншотах используется `docker-compose v1 – pip` модуль. Все команды совместимы с v2, целесообразнее использовать команду `docker compose`

Docker-compose: Панель управления БД

- Выбор образа – mongo-express, версия latest

```
mongo-express:  
  image: mongo-express  
  restart: unless-stopped  
  container_name: "mongo-express"
```

```
docker-compose.yml:  
services:  
  mongodb:  
    mongo-express:  
    ...
```

Docker-compose: Панель управления БД

- Панель управления должна быть доступна. Определим для доступа порт 8081 (На этом же порту запускается панель в контейнере)

```
mongo-express:  
  image: mongo-express  
  restart: unless-stopped  
  container_name: "mongo-express"  
  ports:  
    - 8081:8081
```

```
docker-compose.yml:  
services:  
  mongodb:  
    mongo-express:  
    ...
```

Docker-compose: Панель управления БД

- Для доступа к БД необходимо определить адрес и пару логин/пароль
- Используем линк, чтобы обращаться напрямую к контейнеру – они в одной виртуальной сети

```
mongo-express:
  image: mongo-express
  restart: unless-stopped
  container_name: "mongo-express"
  ports:
    - 8081:8081
  links:
    - mongodb:mongo
  environment:
    ME_CONFIG_MONGODB_ADMINUSERNAME: root
    ME_CONFIG_MONGODB_ADMINPASSWORD: example
    ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
```

```
docker-compose.yml:
services:
  mongodb:
    mongo-express:
      ...
```

Docker-compose: Панель управления БД

- Инструкция `links` предоставляет неявный `depends_on` – инструкцию, определяющую порядок запуска. Таким образом, невозможно сделать двухстороннюю связь через `links` в версии 2.1

```
mongo-express:
  image: mongo-express
  restart: unless-stopped
  container_name: "mongo-express"
  ports:
    - 8081:8081
```

links:

- `mongodb:mongo` # Контейнер `mongodb` доступен под именем `mongo`

environment:

```
ME_CONFIG_MONGODB_ADMINUSERNAME: root
ME_CONFIG_MONGODB_ADMINPASSWORD: example
ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
```

docker-compose.yml:

```
services:
  mongodb:
    mongo-express:
    ...
```

Docker-compose: Панель управления БД

- Используя `docker compose up -d` запускаем сервисы

```
romanutaaa@r9odt-cat:tg-bot [05:46 ] [master] $ docker-compose up -d
mongo is up-to-date
Creating mongo-express ... done
romanutaaa@r9odt-cat:tg-bot [06:11 ] [master] $ docker-compose ps
```

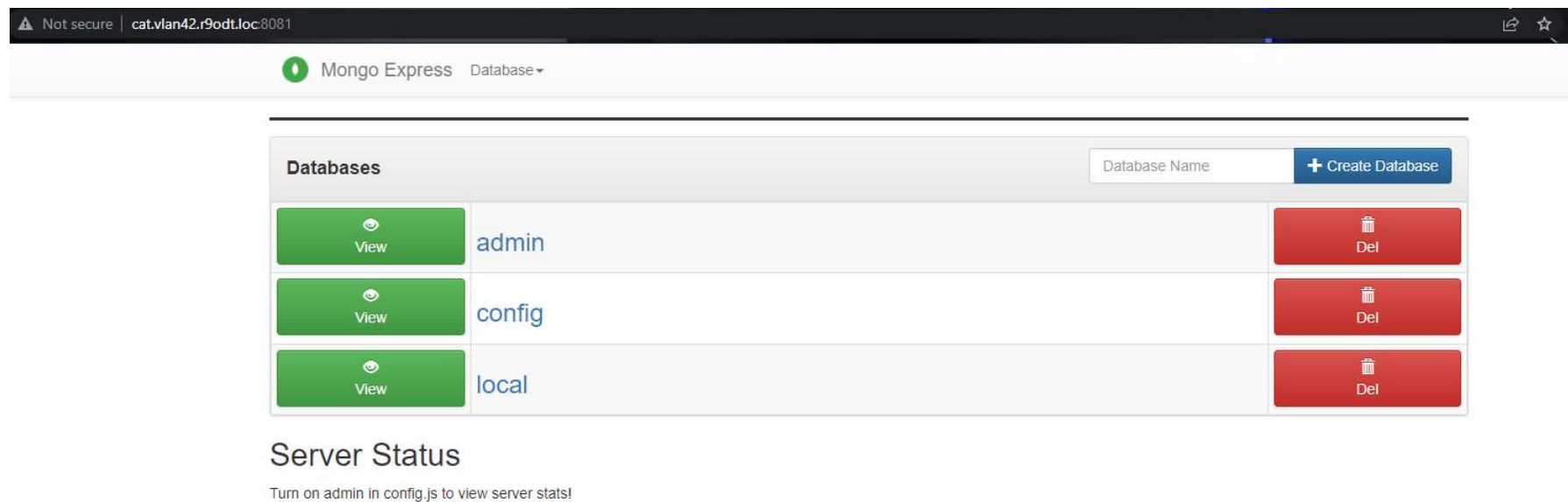
Name	Command	State	Ports
mongo	docker-entrypoint.sh mongod	Up	127.0.0.1:27017->27017/tcp
mongo-express	tini -- /docker-entrypoint ...	Up	0.0.0.0:8081->8081/tcp,:::8081->8081/tcp

```
romanutaaa@r9odt-cat:tg-bot [06:11 ] [master] $
```

Конфигурация контейнера базы данных не изменилась. Добавился сервис mongo-express

Docker-compose: Панель управления БД

- Используя `docker compose up -d` запускаем сервисы



Интерфейс панели управления mongo-express

Docker-compose: Yaml-якорь

- YAML позволяет определить ссылку на секцию конфигурации через & и обратиться к ней через <<: *

```
mongodb:
  image: mongo:4.2.16
  ...
  <<: *logging

mongo-express:
  image: mongo-express
  ...
  <<: *logging
```

```
docker-compose.yml:
x-logging: &logging
  logging:
    options:
      max-size: '32m'
      max-file: '5'
      driver: json-file
  restart: unless-stopped
services:
  mongodb:
  mongo-express:
```


Docker-compose: команда config

- Для просмотра итоговой конфигурации docker-compose используется команда `docker compose config`
- Команда раскрывает все подстановки (В т.ч. переменные среды) и показывает конфигурацию, которая будет непосредственно применена

```
romanutaaa@r9odt-cat:tg-bot [03:58 ] [master] $ docker-compose config
services:
  mongo-express:
    container_name: mongo-express
    environment:
      ME_CONFIG_MONGODB_ADMINPASSWORD: example
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
    image: mongo-express
    links:
      - mongodb:mongo
    logging:
      driver: json-file
      options:
        max-file: '5'
        max-size: 32m
    ports:
      - published: 8081
        target: 8081
    restart: unless-stopped
```

```
mongodb:
  container_name: mongo
  environment:
    MONGO_INITDB_DATABASE: tgbot
    MONGO_INITDB_ROOT_PASSWORD: example
    MONGO_INITDB_ROOT_USERNAME: root
  hostname: r9odt-42
  image: mongo:4.2.16
  logging:
    driver: json-file
    options:
      max-file: '5'
      max-size: 32m
  ports:
    - 127.0.0.1:27017:27017/tcp
  restart: unless-stopped
  volumes:
    - /home/romanutaaa/go/src/github.com/JIexa24/tg-bot/mongo-init.
  version: '2.1'
```

При выполнении `docker compose config` показаны подстановки секции logging

Docker-compose: Приложение

- Перед запуском необходимо собрать docker-образ

```
bot:
  build:
    context: .
    dockerfile: Dockerfile
```

```
docker-compose.yml:
services:
  mongodb:
  mongo-express:
  bot:
    ...
```

Docker-compose: Приложение

- Определим параметры работы – хостнейм, условия перезапуска и имя контейнера

```
bot:
  build:
    context: .
    dockerfile: Dockerfile
  hostname: "tg-bot"
  restart: unless-stopped
  container_name: "bot"
```

```
docker-compose.yml:
services:
  mongodb:
  mongo-express:
  bot:
    ...
```

Docker-compose: Приложение

- Разрешим обращаться к приложению – приложение по умолчанию работает на порту 3000, протокол tcp
- Разрешим доступ до базы данных через сеть контейнера

```
bot:
  build:
    context: .
    dockerfile: Dockerfile
  hostname: "tg-bot"
  restart: unless-stopped
  container_name: "bot"
  ports:
    - "0.0.0.0:3000:3000/tcp"
  links:
    - mongodb:mongo
```

```
docker-compose.yml:
services:
  mongodb:
  mongo-express:
  bot:
    ...
```

Docker-compose: Приложение

- Определим переменные среды окружения – в данном случае, приложение использует их в качестве источника конфигурации

```
bot:
  build:
    context: .
    dockerfile: Dockerfile
  hostname: "tg-bot"
  restart: unless-stopped
  container_name: "bot"
  ports:
    - "0.0.0.0:3000:3000/tcp"
  links:
    - mongodb:mongo
  environment:
    TELGRAM_TOKEN: token
    LOG_LEVEL: debug
    DATABASE_HOST: mongo
```

```
docker-compose.yml:
services:
  mongodb:
  mongo-express:
  bot:
    ...
```

Docker-compose: Приложение

- Если необходимо передать параметр через командную строку, то определяется раздел *command*

```
bot:
  build:
    context: .
    dockerfile: Dockerfile
  command: ["/app/app", "-pprof"]
  hostname: "tg-bot"
  restart: unless-stopped
  container_name: "bot"
  ports:
    - "0.0.0.0:3000:3000/tcp"
  links:
    - mongodb:mongo
  environment:
    TELGRAM_TOKEN: token
    LOG_LEVEL: debug
    DATABASE_HOST: mongo
```

```
docker-compose.yml:
services:
  mongodb:
  mongo-express:
  bot:
    ...
```

Docker-compose: Приложение

- Для сборки, при запуске указывается ключ *build*

```
docker compose up -d --build
```

```
Successfully built d1782a28cb23
Successfully tagged tg-bot_bot:latest
mongo is up-to-date
mongo-express is up-to-date
Creating bot ... done
romanutaaa@r9odt-cat:tg-bot [06:37 ] [master] $ docker-compose ps
```

Name	Command	State	Ports
bot	/app/docker-entrypoint.sh ...	Up	0.0.0.0:3000->3000/tcp, 8082/tcp
mongo	docker-entrypoint.sh mongod	Up	127.0.0.1:27017->27017/tcp
mongo-express	tini -- /docker-entrypoint ...	Up	0.0.0.0:8081->8081/tcp, :::8081->8081/tcp

Приложение собрано, сервисы запущены

Docker-compose: Приложение

- Для просмотра логов используется `docker compose logs`

```
romanutaaa@r9odt-cat:tg-bot [06:39] [master] $ docker-compose logs --tail 5
Attaching to bot, mongo-express, mongo
bot          | 2022-05-31 11:37:47.289 telegram-bot INF Worker 5 started file=worker.go:9 func=(*Bot).worker module=telegram-bot package=lib
bot          | 2022-05-31 11:37:47.289 telegram-bot INF Worker 4 started file=worker.go:9 func=(*Bot).worker module=telegram-bot package=lib
bot          | 2022-05-31 11:37:47.289 telegram-bot INF Worker 6 started file=worker.go:9 func=(*Bot).worker module=telegram-bot package=lib
bot          | 2022-05-31 11:37:47.289 telegram-bot INF Start listening by address: [0.0.0.0:3000] file=httpserver.go:55 func=(*HTTPServer).Listen module=telegram-bot package=httpserver
bot          | 2022-05-31 11:37:47.289 telegram-bot INF Worker 7 started file=worker.go:9 func=(*Bot).worker module=telegram-bot package=lib
mongo-express | GET /public/img/mongo-express-logo.png 304 2.859 ms - -
mongo-express | GET /public/img/gears.gif 304 3.082 ms - -
mongo-express | GET /public/vendor-d1b820f8a9cf3d5a8c6a.min.js 304 3.240 ms - -
mongo-express | GET /public/fonts/glyphicons-halflings-regular.woff2 304 1.322 ms - -
mongo-express | GET /public/database-f7fd2b32a64b0a7d1c38.min.js 200 4.265 ms - 1513
mongo        | 2022-05-31T11:37:47.280+0000 I NETWORK [conn2] received client metadata from 172.19.0.4:58022 conn2: { driver: { name: "mongo-go-driver", version: "v1.9.0" }, os: { type
"amd64" }, platform: "go1.18.1" }
mongo        | 2022-05-31T11:37:47.280+0000 I NETWORK [conn3] received client metadata from 172.19.0.4:58020 conn3: { driver: { name: "mongo-go-driver", version: "v1.9.0" }, os: { type
"amd64" }, platform: "go1.18.1" }
mongo        | 2022-05-31T11:37:47.280+0000 I NETWORK [listener] connection accepted from 172.19.0.4:58024 #4 (4 connections now open)
mongo        | 2022-05-31T11:37:47.280+0000 I NETWORK [conn4] received client metadata from 172.19.0.4:58024 conn4: { driver: { name: "mongo-go-driver", version: "v1.9.0" }, os: { type
"amd64" }, platform: "go1.18.1" }
mongo        | 2022-05-31T11:37:47.288+0000 I ACCESS [conn4] Successfully authenticated as principal tgbot on tgbot from client 172.19.0.4:58024
```

Просмотр лога приложения

Docker-compose: .env и env_file

- Запуск контейнера можно производить с определенным окружением, при этом набор переменных среды можно хранить в файле
- Чтобы загрузить переменные из файла, используется инструкция `env_file`
- Файл конфигурации docker-compose поддерживает переменные среды окружения и может ссылаться на них в описании сервисов.
- Специальный файл `.env` позволяет определить переменные, которые может использовать docker-compose

Docker-compose: .env и env_file

```
docker-compose.yml:  
services:  
  hello:  
    image: ${HELLO_IMAGE}  
    environment:  
      COMPOSE_FILE_VAR: "var in compose file"  
    env_file:  
      - .hello.env
```

```
.env:  
HELLO_IMAGE="hello-world"
```

```
.hello.env:  
COMPOSE_ENV_FILE_VAR="It.s env  
file var"
```

```
"Env": [  
  "COMPOSE_ENV_FILE_VAR=It.s env file var",  
  "COMPOSE_FILE_VAR=var in compose file",  
  "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"  
],  
"Cmd": [  
  "/hello"  
],  
"Image": "hello-world",  
"
```

Внутри контейнера нет переменной HELLO_IMAGE
Образ hello-world

Docker-compose: несколько файлов

```
docker-compose1.yml:  
services:  
  hello1:  
    image: hello-world
```

```
docker-compose2.yml:  
services:  
  hello2:  
    image: hello-world
```

```
(mc) romanutaaa@r9odt-cat:meet [05:18 ] $ docker-compose -f docker-compose1.yml -f docker-compose2.yml config  
services:  
  hello1:  
    image: hello-world  
  hello2:  
    image: hello-world  
version: '2.1'
```

Объединение нескольких compose-файлов

Docker-compose: несколько файлов

docker-compose1.yml:

```
services:
  hello1:
    image: hello-world
    container_name: hello
```

docker-compose2.yml:

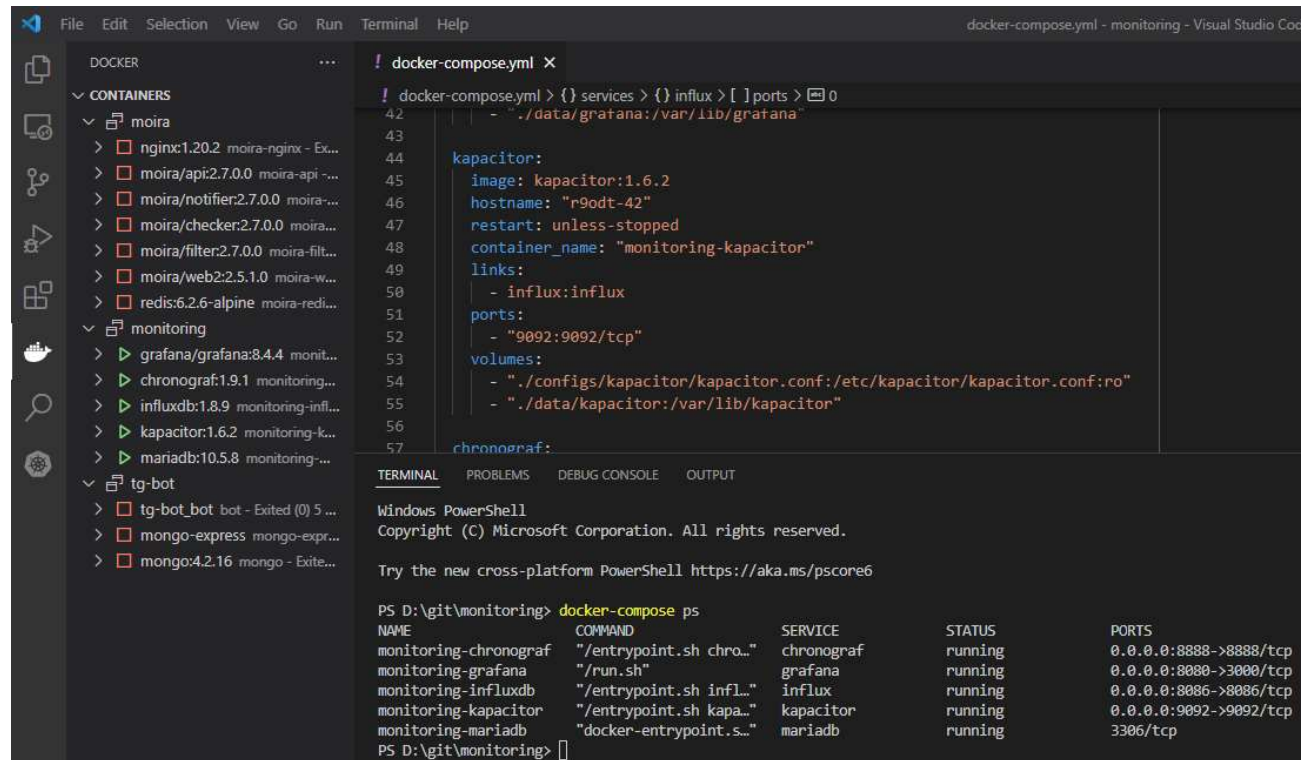
```
services:
  hello1:
    image: nginx
```

```
(mc) romanutaaa@r9odt-cat:meet [05:19 ] $ docker-compose -f docker-compose1.yml -f docker-compose2.yml config
services:
  hello1:
    container_name: hello
    image: nginx
version: '2.1'
```

Переопределение свойств одного файла конфигурации свойствами из другого

Docker-compose на windows

- Docker-compose поставляется вместе с docker-desktop
- Интегрируется с расширением docker для vscode



The screenshot shows the Visual Studio Code interface. On the left, the 'DOCKERS' sidebar lists several containers under 'CONTAINERS'. The main editor area shows the 'docker-compose.yml' file with configuration for 'kapacitor' and 'chronograf'. The bottom terminal window shows the output of the 'docker-compose ps' command in a Windows PowerShell environment.

```
! docker-compose.yml X
! docker-compose.yml > {} services > {} influx > [ ] ports > 0
42 | | - ./data/grafana:/var/lib/grafana
43 |
44 kapacitor:
45   image: kapacitor:1.6.2
46   hostname: "r9odt-42"
47   restart: unless-stopped
48   container_name: "monitoring-kapacitor"
49   links:
50     - influx:influx
51   ports:
52     - "9092:9092/tcp"
53   volumes:
54     - "./configs/kapacitor/kapacitor.conf:/etc/kapacitor/kapacitor.conf:ro"
55     - "./data/kapacitor:/var/lib/kapacitor"
56
57 chronograf:
```

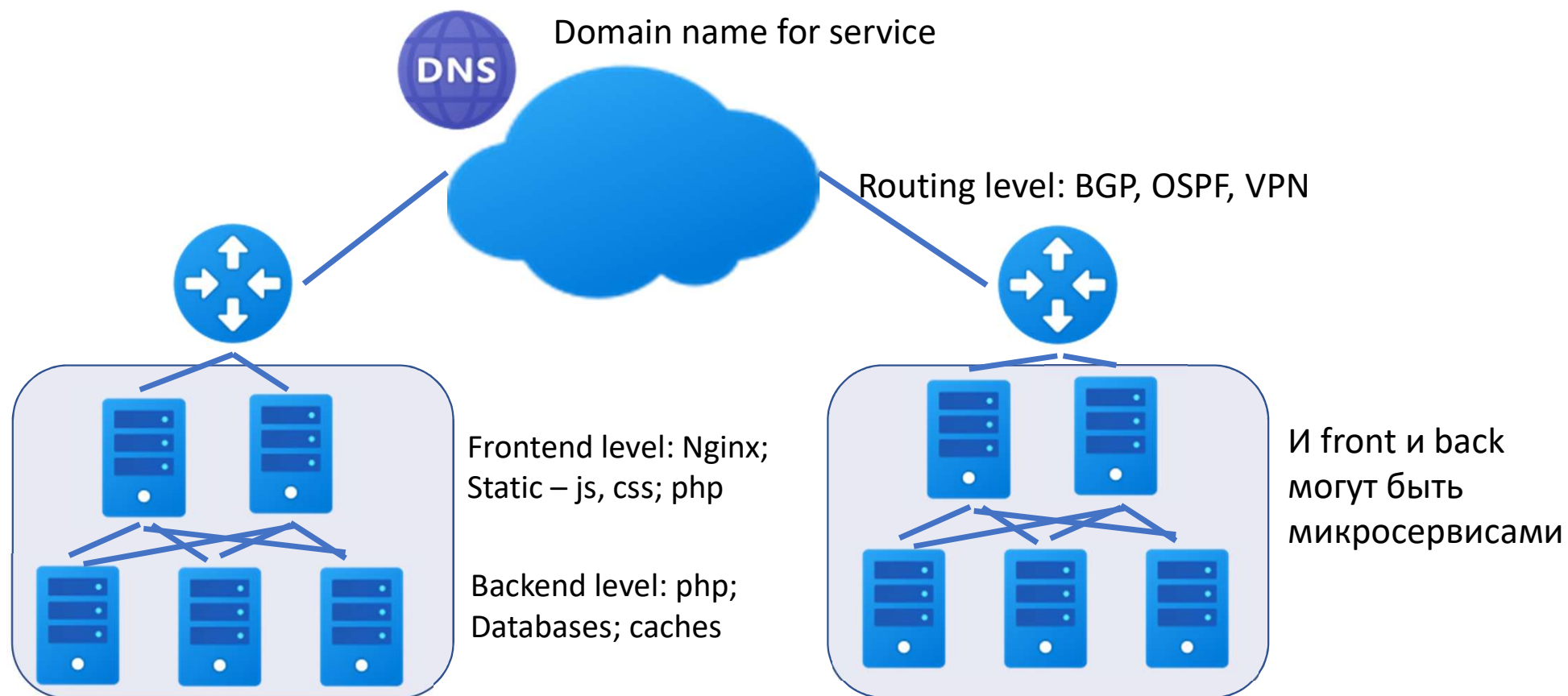
```
TERMINAL  PROBLEMS  DEBUG CONSOLE  OUTPUT
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

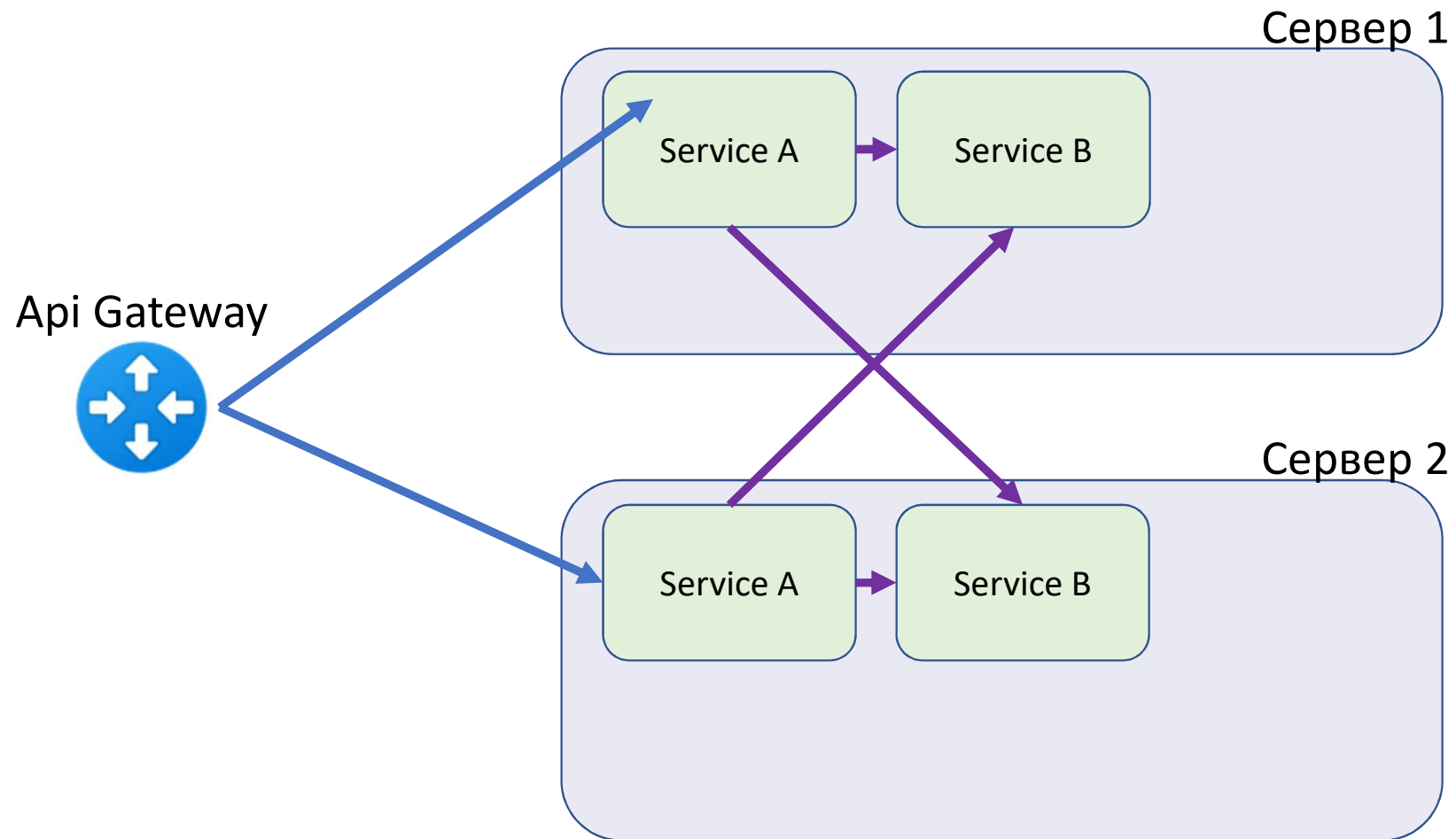
PS D:\git\monitoring> docker-compose ps
NAME                                COMMAND                                SERVICE    STATUS    PORTS
monitoring-chronograf              "/entrypoint.sh chro..."            chronograf running    0.0.0.0:8888->8888/tcp
monitoring-grafana                 "/run.sh"                              grafana    running    0.0.0.0:3000->3000/tcp
monitoring-influxdb                "/entrypoint.sh infl..."            influx     running    0.0.0.0:8086->8086/tcp
monitoring-kapacitor               "/entrypoint.sh kapa..."            kapacitor  running    0.0.0.0:9092->9092/tcp
monitoring-mariadb                 "docker-entrypoint.s..."            mariadb    running    3306/tcp
PS D:\git\monitoring>
```

Просмотр запущенных контейнеров в системе. Вызов docker-compose из консоли PowerShell

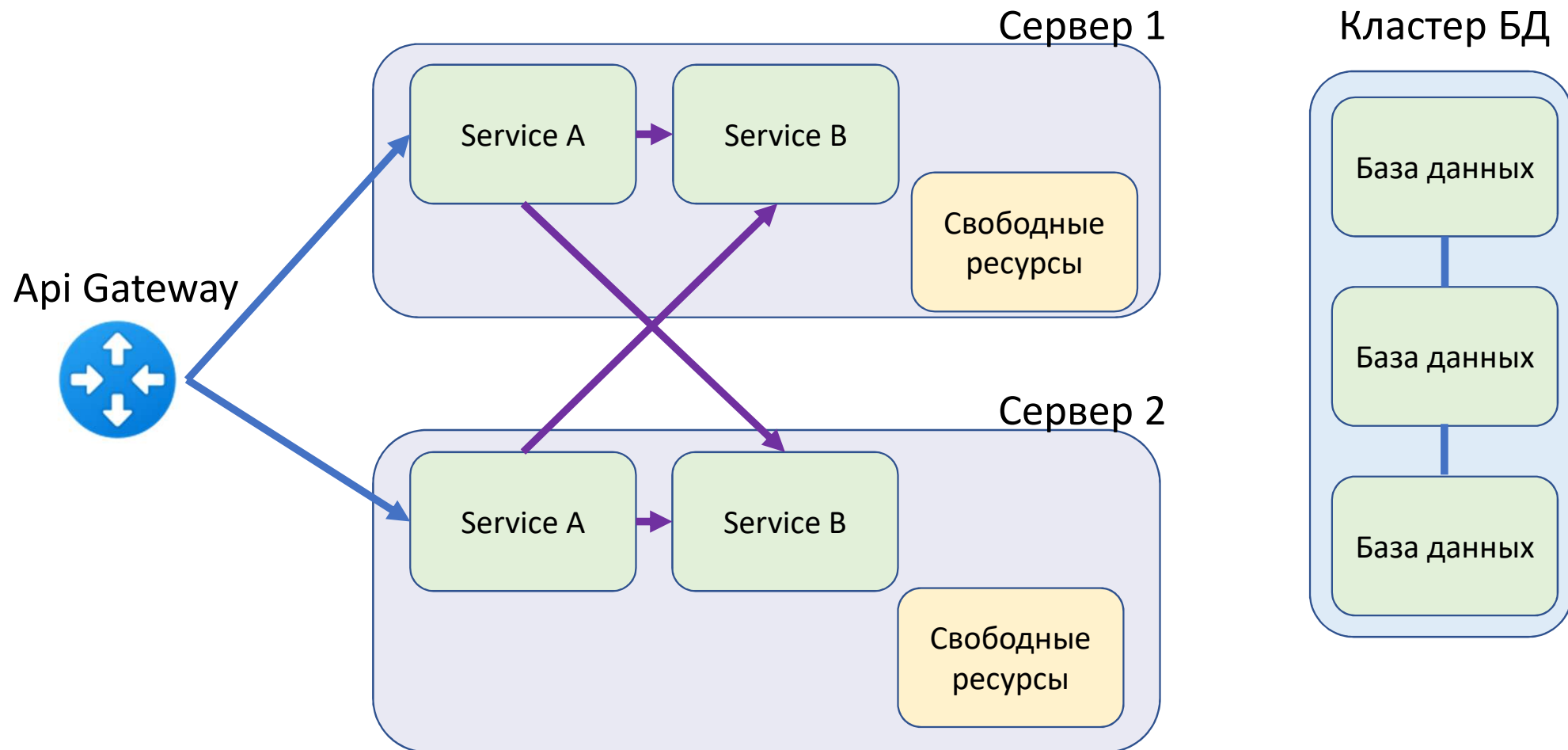
Микросервисы



Микросервисы



Микросервисы

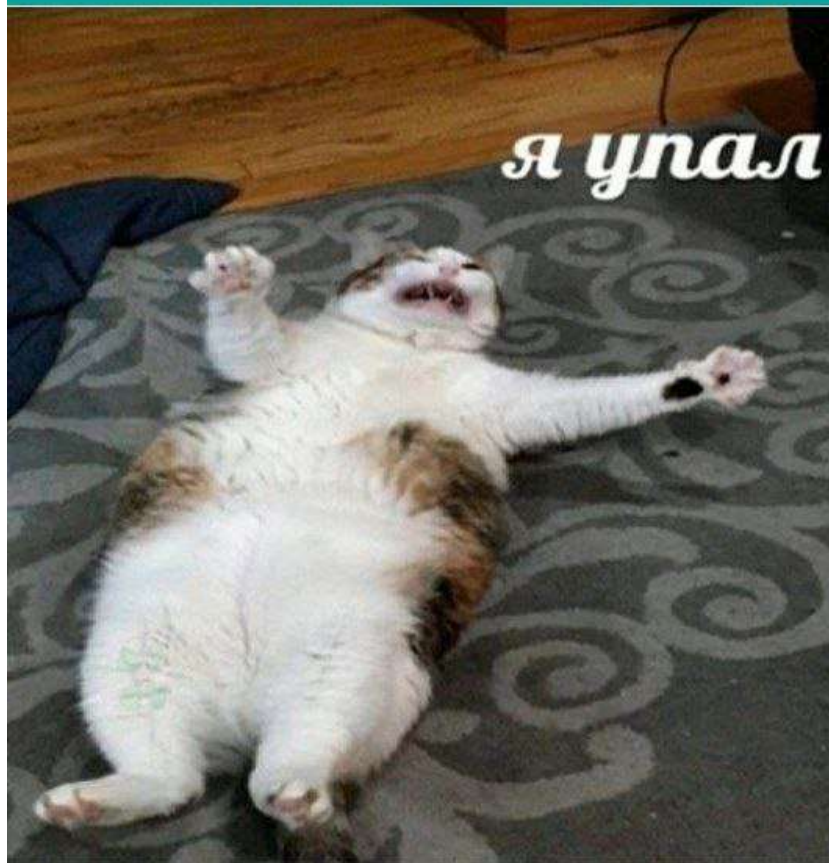


Микросервисы

- Используют асинхронные методы взаимодействия – очереди (kafka, rabbit) и тп
- Stateless
- База данных как правило master и несколько slave хостов – позволяет распределять нагрузку на чтение по репликам БД, что в свою очередь снижает нагрузку на бд и повышает производительность системы в целом

Микросервисы

Монолит



Микросервисы



Live section

Романюта Алексей Андреевич

alexey-r.98@yandex.ru

Кафедра вычислительных систем
Сибирский государственный университет телекоммуникаций и информатики

