СТРПО

Семинар к лабораторным работам №1

Содержание

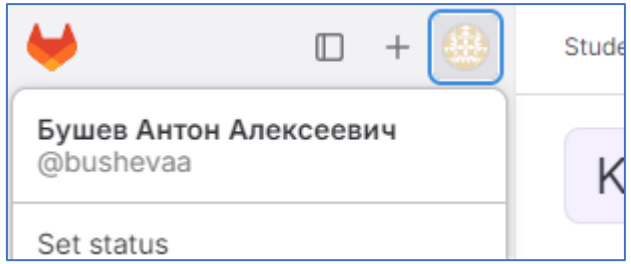
- Темы лабораторных работ
- Инфраструктура
- Golang
- Docker
- Модель Kubernetes

Темы лабораторных работ

- ЛР1: Разработка Golang-микросервисов и публикация Docker-образа в репозитории
- ЛР2: Разработка манифеста кластера Kubernetes и развёртывание конфигурации кластера через сервер системы контроля версий
- ЛР3: Разработка простого фронтенда для микросервиса и развёртывание в кластере Kubernetes

Инфраструктура

- Лабораторные размещаются в Gitlab-репозитории, расположенном на сервере МГТУ им. Н.Э. Баумана https://gitlab.bmstu.ru
- Подключение к Gitlab через VPN МГТУ им. Н.Э. Баумана
- Лабораторные расположены по адресу https://gitlab.bmstu.ru/student/bushevaa/kubernetes
- Для получения задания и прав доступа отправьте в письме преподавателю свой ID (расположен в верхнем левом углу в формате @XXXX)



Инфраструктура

- В группе https://gitlab.bmstu.ru/student/bushevaa/kubernetes/2024 создаются подгруппы на каждую группу студентов
- В группе https://gitlab.bmstu.ru/student/bushevaa/kubernetes/samples расположены примеры проектов
- В проекте https://gitlab.bmstu.ru/student/bushevaa/kubernetes/docker-in-docker-runner-access необходимо получить уровень прав Maintainer для возможности подключения раннера Docker-in-Docker

Golang

- Документация https://go.dev/learn/
- Статическая сборка
- Простая модель языка
- Простая модель многозадачности

Golang

- Переменные и константы
- ```
  var A = 0
  const Pi = 3
  ```
- Условия
- ```
  if true {
  } else if false {
  } else {}
  ```
- Циклы
- ```
  for i := 0; I < 10; i++ {}
  for <condition> {} // while
  ```
  ```

Golang: Встроенные типы

- Неявное приведение типов в Golang отсутствует
- ```
  var i int = 4
  var f float32 = float32(i)
  ```
- ```
  bool = false
  string = “”
  int  int8  int16  int32  int64 = 0
  uint uint8 uint16 uint32 uint64 uintptr
  byte // alias for uint8
  rune // alias for int32
       // represents a Unicode code point
  float32 float64
  complex64 complex128
  ```

Golang: Функции

- Функции в Golang могут возвращать несколько значений
- ```
  func swap(x int, y int) (int, int) {
      return (y, x)
  }
  ```
- ```
  func swap(x int, y int) (xr, yr int) {
      xr = y
      yr = x
      return
  }
  ```

Golang: Модель памяти

- Если компилятор обнаружит возвращение указателя на локальную переменную, то эта переменная будет выделена в куче

Golang: Пакеты

- Текст структурируется пакетами (package)
- Пакет main – пакет точки входа исполняемой программы
- ```
  package main
  import (
      “fmt”
  )
  func main() {
      fmt.Println(“…”)
  }
  ```
  ```

Golang: Пакеты

- Пакеты являются наименьшей единицей модульности
- На уровне пакетов определяется сокрытие имён
- Имена с заглавной буквы открыты для внешних пакетов
- ```
  package pack1
  // Открытое имя
  var A = 0
  // Закрытое имя
  func f(x int, y int) int {
      return x + y
  }
  ```
  ```

Golang: Модули

- Инструментарий Golang поддерживает модули располагаемые в репозиториях Git
- Модули Golang распространяются исходными текстами пакетов
- Пример конфигурационного файла модуля go.mod:
- ```
  module gitlab.bmstu.ru/bushevaa/microservice-example
  go 1.21
  ```
- Пример импорта пакетов:
  ```
  import (
      "gitlab.bmstu.ru/bushevaa/microservice-
  example/internal/timeservice"
      ts "gitlab.bmstu.ru/bushevaa/microservice-
  example/internal/timesource“
  )
  ```
  ```

Golang: Структуры

- Структуры
- ```
  type StructName struct {
      OpenX int
      closeY int
  }
  ```
- Инициализация структур
- ```
  var s = StructName{0, 1}
  var s = StructName{OpenX: 0, closeY: 1}
  ```

Golang: ООП

- `type Point struct {}`
- ```
  func f(p Point, int) int {…}
  func main() {
      p := Point{}
      a := f(p, 1)
  }
  ```
- ```
  func (p Point) f(int) int {…}
  func main() {
      p := Point{}
      a := p.f(1)
  }
  ```

Golang: Интерфейсы

- Интерфейс – указатель на значение и набор функций
- ```
  type OpenClose interface {
      open()
      close()
  }
  ```
- ```
  type OpenCloseImpl struct {}
  func (oci *OpenCloseImpl) open() {}
  func (oci *OpenCloseImpl) close() {}
  ```
- ```
  func main() {
      var oci *OpenCloseImp = nil
      var oc OpenClose = oci
      oc.open()
  }
  ```
  ```

Golang: Обработка ошибок

- error – встроенный интерфейс ошибок
- ```
  type error interface {
      Error() string
  }
  ```
- Ошибки принято возвращать вторым значением
- ```
  func maybe_error() (int, error) {…}
  func main() {
      a, err := maybe_error()
      if err != nil {…}
  }
  ```

Golang: Срезы

- Массивы фиксированной длины
- `var s_arr [4]string`
  `s_arr[2] = “s”`
- Срезы (slice) представляют собой структуру из указателя на первый элемент и количества элементов
- `var s_slice []string`
  `s_slice = s_arr[0:5]`
  `s_slice = [3]string{“”, “”, “”}`
  `s_slice = make([]string, <length>, <capacity>)`
  `var ss_slice [][]string = [][]string{`
  `[]string{…}, []string{…}, []string{…} }`

Golang: Срезы

- Длина среза
- `var s_len = len([]int{1, 2}) // == 2`
- Добавление в конец среза
- `s := append([]int{}, 1) // == []int{1}`
- Перебор среза
- `for index, value := range s {}`

Golang: Словари

- Словари (map)
- ```
  type Point struct {}
  var str_point_dict map[string]Point
  str_point_dict = make(map[string]Point)
  str_point_dict[“”] = Point{}
  str_point_dict = map[string]Point { “”: Point{},
  “1”: {} }
  ```
- Удаление из словаря
- `delete(str_point_dict, “”)`
- Проверка вхождения
- `elem, is_in := str_point_dict[“3”]`
  ```

Golang: Функции Высшего Порядка

- Функции высшего порядка
- ```
  func f(fn_var func(int,int) int) {
      var i = fn_var(0, 2)
  }
  ```
  ```

Golang: Тестирование

- Для тестирования файла file.go нужно создать file_test.go:
- ```
  package <тот же, что и у file.go>
  import “testing”
  func TestXXX (t *testing.T) {
      ...
      t.ErrorF(“%v”, x)
  }
  ```
- Вызвать команду go test file_test.go или go test ./…
  ```

Golang: Горутины

- Горутины
- ```
  func f() {…}
  func main() {
      go f()
  }
  ```
- Каналы, буферизированные каналы
- ```
  var c chan int = make(chan int)
  c = make(chan int, 2)
  ```
- Получение, отправка, закрытие
- ```
  var a int
  var ok bool
  a, ok = <-c
  c <- a
  close(c)
  ```
  ```

Docker

- Документация https://docs.docker.com
- Литература https://github.com/ivanporty/cloud-docker-k8s-book

Docker: Контейнеризация

- Изоляция
- Неизменяемость
- Воспроизводимость
- Dependency Hell solution
- Совмещение разработки и развёртывания
- Легковесность на уровне процесса операционной системы

Docker: Утилита docker

- `docker {container} create`
- `docker {container} start`
- `docker {container} stop`
- `docker {container} rm`

Docker: Dockerfile

- Dockerfile – сценарий сборки образа контейнера
- Образы представляют собой набор слоёв: результатов работы тех или иных сценариев
- `FROM`
- `WORKDIR`
- `COPY`
- `RUN`
- `CMD`
- Сборка образов
- `docker build <path> -t <tag>`

Docker: Volumes

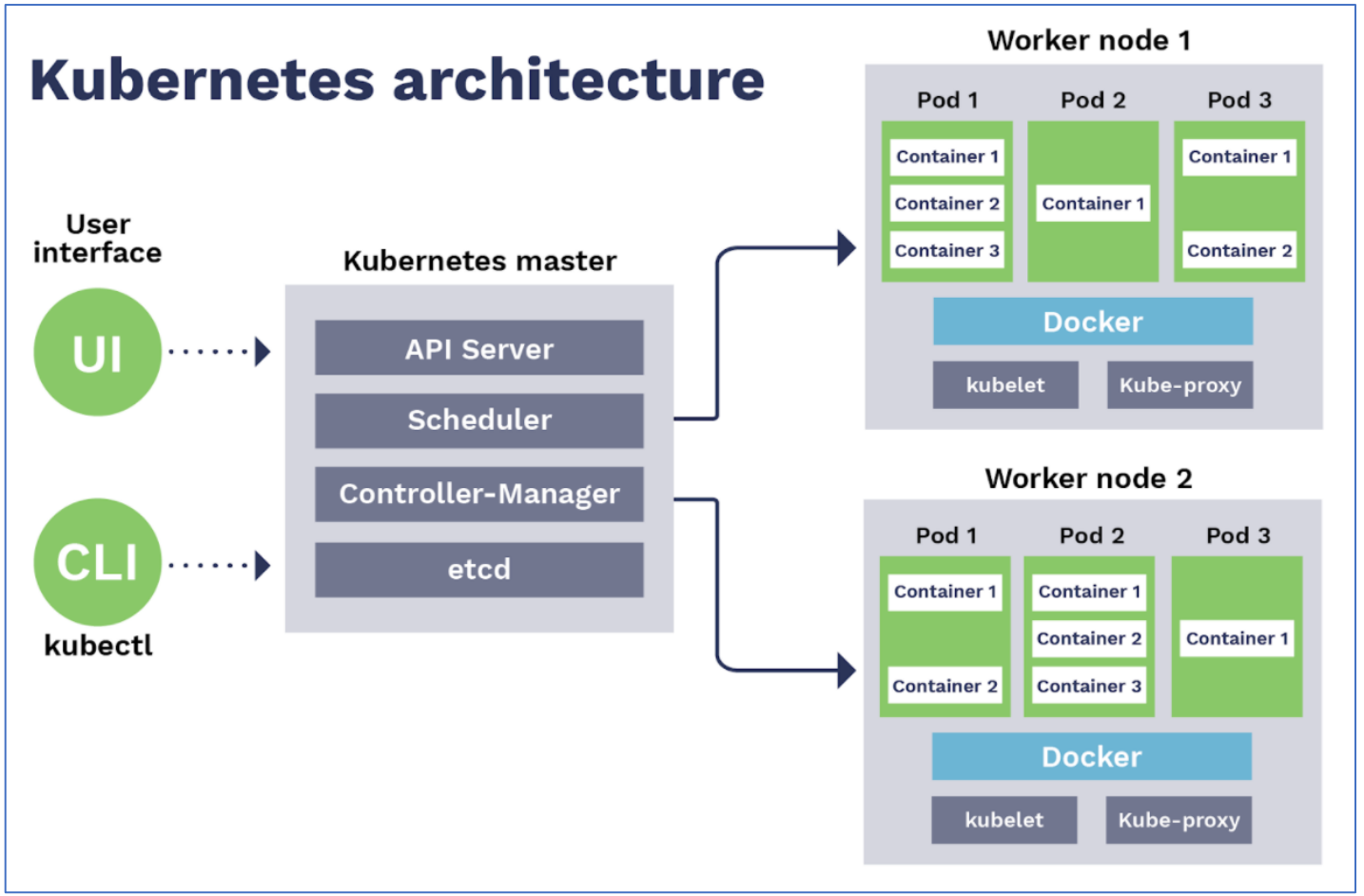
- Том (volume) – технология Docker, позволяющая размещать в файловой системе постоянное хранилище для контейнеров
- Подключение томов происходит на этапе создания контейнера

Модель Kubernetes

- Документация https://kubernetes.io
- Литература https://github.com/ivanporty/cloud-docker-k8s-book

Модель Kubernetes

- Кластер
- Управляющий узел
- Рабочий узел
- Отсек
- Развёртывание
- Служба

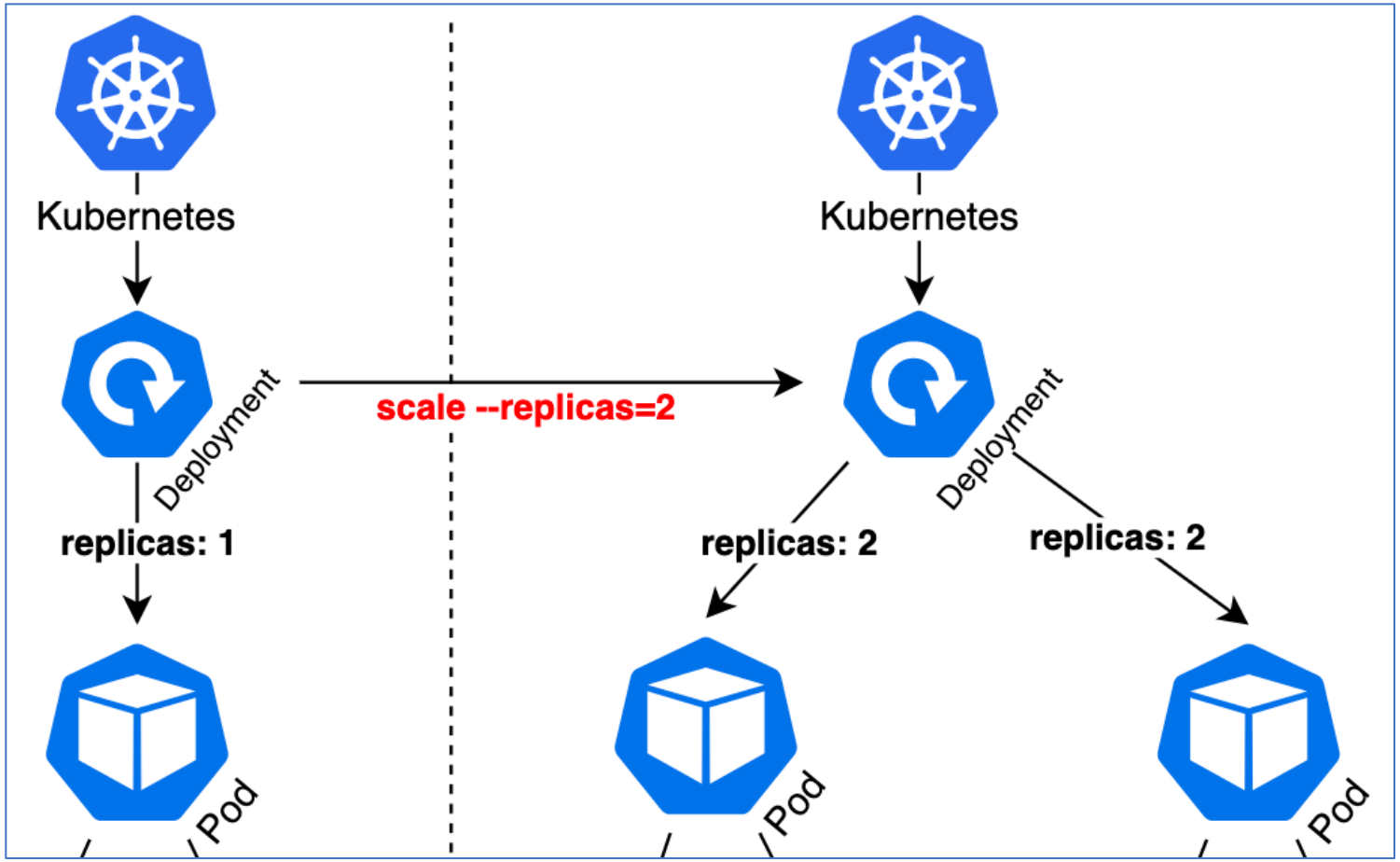


Модель Kubernetes: Отсек

- Наименьшая логическая единица кластера
- Как правило соотношение контейнеров к отсекам 1:1
- Отсеки изолированы так же как контейнеры

Модель Kubernetes: Развёртывание

- Масштабирование отсеков



Модель Kubernetes: Служба

- Доступ к развёртыванию