МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ

по дисциплине «Технологии контейнеризации»

для студентов направления подготовки
09.03.02 «Информационные системы и технологии»
Профиль «Прикладное программирование в интеллектуальных информационных системах»

Ставрополь, 2026

Оглавление

ВВЕДЕНИЕ

В методических указаниях к лабораторным работам по дисциплине «Технологии контейнеризации» студенты учатся работать с механизмами, обеспечивающими работу контейнеров в GNU/Linux, основами работы с контейнерами при помощи Docker и Podman, а также системой оркестрирования контейнеров Kubernetes. Помимо этого, книга знакомит с особенностями одного из самых популярных дистрибутивов Kubernetes – OpenShift (OKD).

ЛАБОРАТОРНАЯ РАБОТА 1. ОСНОВЫ КОНТЕЙНЕРИЗАЦИИ

1. Цель и содержание

Цель лабораторной работы: изучить основы технологии контейнеризации.

Задачи лабораторной работы:

– ознакомиться с основной терминологией контейнеризации;

– изучить основные доступные инструментальные средства;

– исследовать архитектуру контейнеров.

2. Теоретическое обоснование

2.1. История Docker

Готовые к промышленному применению контейнеры на открытых технологиях в GNU/Linux появились позже, чем механизм Jails во FreeBSD (2000 год) или Zones в Solaris (2005 год). Долгое время самой известной в GNU/Linux контейнерной технологией оставался коммерческий продукт Virtuozzo (с 2001 года) компании, в разные годы называвшейся Swsoft, Parallels и Odin. Часть кода Virtuozzo была открыта в 2005 году в виде проекта OpenVZ. Примерно с того же времени развивается проект Linux-VServer, требующий патчей к ядру Linux (нулевая версия появилась в 2001 году).

В 2006 году появился механизм cgroups (от англ. control groups – ресурсные, или контрольные, группы) как редизайн существующей технологии cpuset (подробнее о cpuset в блоге автора книги в заметке 2009 года). В 2008 году появились пользовательские пространства имен, которые

стали еще одной составной частью контейнеров GNU/Linux. Про контрольные группы и пространства имен мы поговорим далее в книге.

В 2008 году инженерами IBM был инициирован проект LXC, который долгое время в первую очередь ассоциировался с открытыми контейнерами GNU/Linux в противовес проприетарному Virtuozzo. Первая версия LXC вышла только в 2014 году, получив в том числе поддержку системы мандатного контроля доступа (MAC) SELinux.

В июле 2015 года Linux Foundation анонсировала запуск нового проекта Open Container Project (OCP), который призван установить общие стандарты и обеспечить фундамент совместимости для продолжения развития контейнерных решений без дальнейшей фрагментации этого направления. Инициативу OCP поддержали многие крупные компании и организации, среди которых можно упомянуть Amazon Web Services, Apcera (в дальнейшем компания поглощена компанией Ericsson), Cisco, CoreOS (поглощена компанией Red Hat), EMC, Google, HP, IBM, Intel, Microsoft, Red Hat (теперь принадлежит IBM), VMware и др. В качестве основы Open Container Project выступили в значительной степени наработки проекта и компании Docker.

На настоящий момент OCI поддерживает две спецификации:

- спецификация среды выполнения (runtime-spec);
- спецификация образов контейнеров (image-spec).

В июле 2015 года была выпущена первая версия системы оркестрации контейнеров Kubrnetes.

2.2. Архитектура Docker

Docker использует клиентсерверную архитектуру и состоит из клиента – утилиты docker, которая обращается к серверу при помощи REST API, и демона в операционной системе GNU/Linux (dockerd). Хотя Docker работает

и в отличных от GNU/Linux операционных системах, в этом курсе они не рассматриваются.

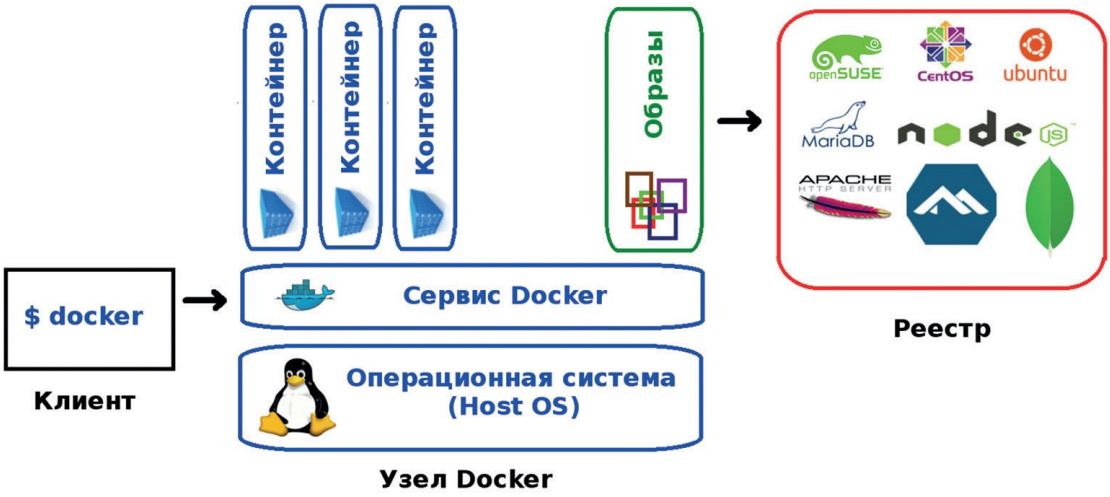

Рисунок 1.1 – Архитектура контейнеров Docker в GNU/Linux

Основные компоненты Docker:

1. **Контейнеры** – изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения. Проще всего дать определение контейнеру Docker как запущенному из образа приложению. Кстати, именно этим идеологически и отличается Docker, например, от LXC (Linux Containers), хотя они используют одни и те же технологии ядра Linux. Разработчики проекта Docker исповедуют принцип: один контейнер – это одно приложение;

2. **Образы** – доступные только для чтения шаблоны приложений. Поверх существующих образов могут добавляться новые уровни образов, которые совместно представляют файловую систему, изменяя или дополняя предыдущий уровень. Обычно новый образ создается либо при помощи сохранения уже запущенного контейнера в новый образ поверх существующего, либо при помощи специальных инструкций для утилиты Dockerfile. Для разделения различных уровней контейнера на уровне

файловой системы могут использоваться UnionFS, aufs, btrfs, vfs, OverlayFS и Device Mapper;

3. **Реестры** (registry), содержащие репозитории (repository) образов, – сетевые хранилища образов. Могут быть как приватными, так и общедоступными. Самым известным реестром является Docker Hub [3]. В книге также упоминаются репозитории, содержащие rpmпакеты. С ними работают команды yum и dnf. Не путайте репозитории, служащие для установки пакетов операционной системы, и репозитории Docker.

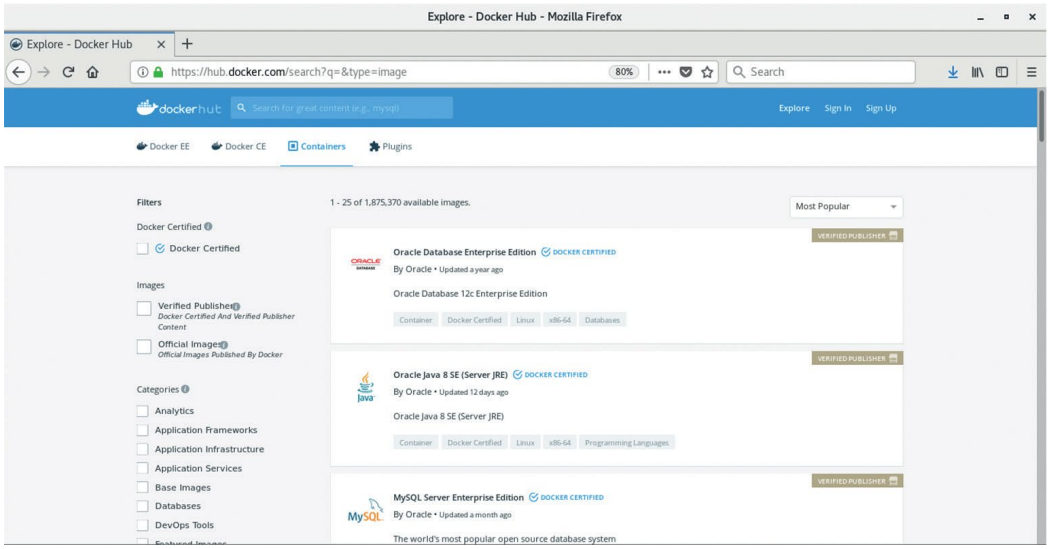

Рисунок 1.2 – Реестр образов Docker Hub

Механизм контрольных групп (cgroups) предоставляет инструмент для тонкого контроля над распределением, приоритизацией и управлением системными ресурсами. Контрольные группы реализованы в ядре Linux. В современных дистрибутивах управление контрольными группами реализовано через systemd, однако сохраняется возможность управления при помощи библиотеки libcgroup и утилиты cgconfig. Основные иерархии контрольных групп (их также называют контроллерами) перечислены ниже:

- **blkio** – задает лимиты на операции вводавывода и на доступ к блочным устройствам;

- **cpu** – используя планировщик процессов, распределяет процессорное время между задачами;

- **cpuacct** – создает автоматические отчеты по использованию ресурсов центрального процессора. Работает совместно с контроллером cpu, описанным выше;

- **cpuset** – закрепляет за задачами определенные процессоры и узлы памяти;

- **devices** – регулирует доступ задач к определенным устройствам;

- **freezer** – приостанавливает или возобновляет задачи;

- **memory** – устанавливает лимиты и генерирует отчеты об использовании памяти задачами контрольной группы;

- **net_cls** – осуществляет тегирование сетевых пакетов идентификатором класса (classid). Это позволяет контроллеру трафика (команда tc) и брандмауэру (iptables) учитывать данные теги при обработке трафика;

- **perf_event** – позволяет производить мониторинг контрольных групп при помощи утилиты perf;

- **hugetlb** – дает возможность использовать виртуальные страницы памяти большого размера и применять к ним лимиты.

Пространства имен, в свою очередь, контролируют не распределение ресурсов, а доступ к структурам данных ядра. Фактически это означает изоляцию процессов друг от друга и возможность иметь параллельно «одинаковые», но не пересекающиеся друг с другом иерархии процессов, пользователей и сетевых интерфейсов. При желании разные сервисы могут иметь даже свои собственные loopbackинтерфейсы.

Примеры пространств имен, используемых Docker:

- **PID, Process ID** – изоляция иерархии процессов;

- **User** – изоляция идентификаторов пользователей (UID) и групп (GID);

- **NET, Networking** – изоляция сетевых интерфейсов;

- **IPC, InterProcess Communication** – управление взаимодействием между процессами;

- **MNT, Mount** – управление точками монтирования;

- **UTS, Unix Timesharing System** – изоляция ядра, идентификаторов версии, имени хоста и доменного имени NIS.

3. Методика и порядок выполнения работы

1. Установите Docker.

2. Запустите сервис Docker.

3. Проверьте статус сервиса Docker.

4. С использованием команды просмотрите информацию о сервисе и окружении:

```
docker info
```

4. Вопросы для защиты работы

1. Назовите отличия использования контейнеров по сравнению с виртуализацией (укажите все правильные ответы).

A. Меньшие накладные расходы на инфраструктуру

B. Время старта приложений больше

C. Невозможность запуска GNU/Linux и Windows-приложений
на одном хосте

D. Обязательное использование гипервизора KVM

2. Назовите основные компоненты Docker (укажите все правильные ответы).

A. Гипервизор

B. Контейнеры

C. Образы виртуальных машин

D. Реестры

3. Какие технологии используются для работы с контейнерами в GNU/Linux (укажите все правильные ответы)?

A. Пространства имен (Linux Namespaces)

B. Подключаемые модули аутентификации (PAM)

C. Контрольные группы (cgroups)

D. Аппаратная поддержка виртуализации

ЛАБОРАТОРНАЯ РАБОТА 2. РАБОТА С КОНТЕЙНЕРАМИ DOCKER

1. Цель и содержание

Цель лабораторной работы: научиться работать с снрвисом Docker с использованием базовых команд.

Задачи лабораторной работы:

– научиться использовать команды запуска, изоляции;

– научиться управлять состоянием контейнеров;

– научиться обмениваться с контейнером данными по сети.

2. Теоретическое обоснование

Изучите основные комманды Docker.

3. Методика и порядок выполнения работы

Продемоснтрируйте использование команд для сервиса Docker. Основные команды, которые необходимо использовать: команды запуска, изоляции; команды управления состоянием контейнеров; команды для обмена данными с контейнером данными по сети; просмотр информации о контейнере; команда подключения к контейнеру постоянного хранения; команда публикации контейнера в реестре Docker Hub; команды экспорта/импорта контейнеров; запуск контейнеров с docker и systemd.

4. Вопросы для защиты работы

1. Как найти образ на Docker Hub?

A. docker search <имя>

B. docker find <имя>

C. docker retrieve <имя>

D. docker <имя>

2. Как удалить образ контейнера из локального кеша?

A. docker image delete <имя>

B. docker rmi <имя>

C. docker kill <имя>

D. docker delete <имя>

3. Какие утверждения верны для тега образа (укажите все правильные ответы)?

A. Нельзя задать более одного тега для образа

B. Тег может состоять из нескольких частей

C. Как минимум должен быть один тег у любого образа

4. При помощи команды docker exec -it <имя> bash можно открыть интерактивный сеанс работы с bash для любого запущенного образа.

A. Верно

B. Не верно

ЛАБОРАТОРНАЯ РАБОТА 3. СОЗДАНИЕ КОНТЕЙНЕРОВ С DOCKERFILE

1. Цель и содержание

Цель лабораторной работы: познакомимся с файлом инструкций для сборки контейнеров Dockerfile.

Задачи лабораторной работы:

– упаковка самостоятельно разработанных программ;

– сборка с Dockerfile.

2. Теоретическое обоснование

Dockerfile – это текстовый файл, в котором последовательно записаны команды по созданию образа контейнера. Как правило, одна строка – это одна команда с соответствующими аргументами. Допускаются пустые строки и строки с комментарием. Комментарии начинаются с символа «решетки» – #. Порядок имеет значение, и команды будут исполняться поочередно от первой к последней. При этом каждая инструкция в Dockerfile выполняется независимо.

Первой значимой инструкцией в Dockerfile должна быть инструкция FROM, которая указывает, из какого базового образа создается ваш образ. Мы не будем рассматривать создание базовых образов «с нуля». Для подавляющего большинства задач предпочтительнее использовать готовые базовые образы. Однако мы рассмотрим специальные инструкции, которые нужны, когда наш образ, созданный из базового, используют в качестве базового для «дочернего» образа.

3. Методика и порядок выполнения работы

Синтаксис Dockerfile близок к синтаксису конфигурационных файлов .ini. На каждую строчку приходится одна инструкция. Инструкции пишутся капсом, а их значения отделяются пробелом.

Dockerfile имеет следующую логику заполнения:

1. Первой инструкцией всегда идёт FROM с указанием родительского образа. Например, FROM python:latest.

2. Инструкция RUN может принимать конвейер команд Linux, чтобы не создавать лишние слои. Например, RUN apt-get update && apt-get install python3-pip -y && pip install --upgrade pip && pip install pipenv.

3. Инструкция WORKDIR устанавливает рабочий каталог контейнера. Например, WORKDIR /usr/src/app/. Последующие команды RUN, CMD, ENTRYPOINT наследуют привязку WORKDIR.

4. Завершающей инструкцией всегда идёт CMD. Например, CMD ["python", "web_interface.py"]. CMD наследует привязку к WORKDIR, поэтому web_interface.py будет запущен из папки /usr/src/app/.

Приведем пример несложного Dockerfile, и на его примере разберем логику взаимодействия инструкций между собой.

```
FROM python:latest
RUN apt-get update && apt-get install python3-pip -y && pip install --upgrade pip && pip install pipenv
RUN mkdir -p /usr/src/app/
WORKDIR /usr/src/app/
COPY . /usr/src/app/
EXPOSE 5000
RUN pip install --no-cache-dir -r requirements.txt
CMD ["python", "web_interface.py"]
```

Dockerfile, как и положено, открывается инструкцией FROM с указанием родительского образа. Выбрать подходящий образ можно на hub.docker.com или с помощью команды docker search имя образа. Например, docker search python.

Далее следуют несколько команд RUN. Поскольку каждая инструкция создает новый слой – хорошей практикой считается писать конвейеры логически связанных команд в одной инструкции RUN. Например, мы поместили команды обновления пакетов apt, установку пакетного менеджера pip и его обновление в одну инструкцию. Для комбинирования команд используется логический оператор «И», обозначающийся как &&.

После серии инструкций RUN идет инструкция WORKDIR, которая устанавливает рабочую директорию контейнера. Все последующие команды, работающие с привязками к директории, такие как RUN, CMD, ENTRYPOINT будут выполняться исходя из установленной директории.

Далее идёт инструкция EXPOSE, которая выражает намерение открыть заданный порт. Инструкция сама по себе не открывает порт без применения команды docker run с ключом -P. Если нужно «повесить» контейнер на определённый внешний порт с переадресацией во внутренний порт контейнера — применяется ключ -p с указанием внутреннего и внешнего порта через «:». Например, docker run -p 5000:8080.

Предпоследней идет опять инструкция RUN. Мы уже рассматривали её выше, однако здесь она исполняется с зависимостью от WORKDIR. Инструкция WORKDIR в качестве рабочей директории установила /usr/src/app/, поэтому текущая инструкция RUN будет выполнять команду pip install --no-cache-dir -r requirements.txt из директории установленной инструкцией WORKDIR.

Завершающая инструкция в нашем Dockerfile — CMD с командой ["python", "web_interface.py"]. Опять же заметим, что здесь CMD связана с WORKDIR и скрипт web_interface.py будет выполнен из директории

/usr/src/app/. Важно запомнить, что CMD всегда идет последней и должна быть в единственном экземпляре. Приведем таблицу инструкций Dockerfile с примерами команд и опций:

Инструкция	Описание	Пример использования	Комментарий
FROM	Задает базовый образ. Все последующие инструкции создают слои поверх родительского образа.	FROM python:latest FROM debian:wheezy	Быстрее всего можно найти образ с нужным тегом на Docker hub.
RUN	Выполняет команду внутри контейнера и сохраняет результат.	RUN mkdir /usr/src/app/ RUN apt-get update && apt-get install python3-pip -y	RUN может исполнять конвейер команд с логическими операторами && и ||.
COPY	Копирует файлы и папки из текущей директории, где находится пользователь в указанную директорию в контейнере	COPY . /usr/src/app/	COPY считывает позицию пользователя на хосте, поэтому первым аргументом идет «.».
ADD	Копирует файлы и папки из текущей позиции пользователя, скачивает файлы по URL и работает с tar-архивами.	ADD https://1cloud.ru/archive/api_config.ini /usr/src/app/	Официальная документация не рекомендует применять ADD. Для скачивания по URL можно использовать RUN с CURL или WGET, а для копирования — COPY.
CMD	Выполняет команду с указанными аргументами во время запуска контейнера.	CMD ["python", "web_interface.py"]	CMD должна быть одна в конце Dockerfile. CMD может вызывать исполняемый файл — .sh Аргументы docker run переопределяют CMD. Если в Dockerfile нет CMD, обязательно должна быть инструкция ENTRYPOINT.
ENTRYPOINT	Похожа на CMD, но при запуске контейнера не	ENTRYPOINT ["python",	ENTRYPOINT может использоваться

Инструкция	Описание	Пример использования	Комментарий
	переопределяется в отличие от CMD.	"web_interface.py"]	совместно с CMD.
ENV	Задает переменные среды внутри образа, на которые могут ссылаться другие инструкции.	ENV ADMIN="ivan"	ENV часто применяется для передачи информации в контейнеризированное приложение через переменные среды.
ARG	Задает переменные, значение которых передается докером во время сборки образа.	ARG maintainer=ivan	В отличие от ENV-переменных, ARG-переменные недоступны во время выполнения контейнера.
WORKDIR	Устанавливает рабочую директорию контейнера.	WORKDIR /usr/src/app/	Последующие инструкции CMD, RUN, ENTRYPOINT наследуют привязку к директории установленной WORKDIR.
VOLUME	Создает и подключает постоянный том хранения данных.	VOLUME /data_cont_1	Просмотреть существующие тома можно командой docker volume ls. К контейнеру можно подключить существующий том, для этого достаточно указать уже существующий том.
EXPOSE	Указывает планируемый рабочий порт у контейнера. Инструкция сама по себе не открывает порт. Чтобы использовался указанный в EXPOSE порт — нужно указать docker run -P при запуске контейнера.	EXPOSE 5000	Если требуется пробросить и сопоставить разные порты внутри и снаружи контейнера используется docker run -p внутренний порт:внешний порт
LABEL	Добавляет метаданные в образ.	LABEL maintainer="katkov_iva	Обычно LABEL содержит информацию

Инструкция	Описание	Пример использования	Комментарий
		n"	об авторе образа.

Продемонстрируйте использование рассмотренных команд Dockerfile.

4. Вопросы для защиты работы

1. Что такое базовый образ?

A. Образ дистрибутива с установленными базовыми библиотеками

B. Образ, из которого создается новый образ

C. Образ, в котором отсутствуют конфигурационные файлы программного обеспечения

D. Образ базы данных

2. Какое утверждение верное?

A. В Dockerfile обязательно наличие инструкции ENTRYPOINT

B. Значение ENTRYPOINT по умолчанию /bin/bash c

C. В Dockerfile может быть несколько инструкций CMD

3. Инструкция EXPOSE автоматически делает доступным указанный порт контейнера по сети.

A. Верно

B. Не верно

ЛАБОРАТОРНАЯ РАБОТА 4. РАБОТА С КОНТЕЙНЕРАМИ БЕЗ ДВИЖКА DOCKER

1. Цель и содержание

Цель лабораторной работы: изучить альтернативные инструменты взаимодействия с Docker.

Задачи лабораторной работы:

– научиться использовать CRI-O;

– научиться использовать podman;

– научиться использовать buildah и skopeo.

2. Теоретическое обоснование

Проект CRI-O был инициирован для создания альтернативной Docker «облегченной» среды исполнения контейнеров для системы оркестрации Kubernetes. Мы будем обсуждать Kubernetes позже, но сейчас, для того чтобы перебросить «мост» от среды запуска контейнеров к оркестратору и определить роль рассматриваемых утилит, необходимо кратко коснуться такого понятия, как интерфейс среды исполнения контейнеров – Container Runtime Interface (CRI).

Изначально единственной средой в контексте Kubernetes являлся Docker. Затем к нему прибавился rkt от проекта CoreOS. Параллельно компания Microsoft начала работать над своей средой исполнения контейнеров в Windows. Для унификации взаимодействия между оркестратором контейнеров и средой исполнения и необходим стандартный интерфейс (CRI). CRI-O расшифровывается как OCI-compliant Container Runtime Interface (совместимый с CRI). Рассматриваемые утилиты предназначены заменить интерфейс командной строки docker при работе с

CRI-O, но также могут использоваться и без среды исполнения контейнеров. Перечислим их.

Утилита podman позволяет запускать контейнеры и управлять образами контейнеров. Она поддерживает большинство опций команды docker. Главное отличие в том, что podman не требует демона docker или какойлибо иной запущенной среды выполнения контейнеров. Также данная утилита совместима по структуре каталогов с CRI-O и позволяет работать не только с отдельными контейнерами, но и с podмодулями (связанными наборами контейнеров).

Утилита skopeo предназначена для работы с реестрами образов и самими образами.

Утилита buildah поддерживает синтаксис Dockerfile и представляет собой альтернативу команды docker build. Опять же, она не требует среды исполнения контейнеров.

3. Методика и порядок выполнения работы

1. Продемоннстрировать использование podman для запуска контейнеров, запуска pod-модулей. Продемонстрировать запуск контейнеров с podman и system.

2. Продемонстрировать использование buildah для создания образов контейнеров.

3. Продемонстрируйте использование skopeo для работы с образами.

4. Вопросы для защиты работы

1. Какие из утилит не требуют привилегий root (укажите все правильные ответы)?

A. podman

B. docker

C. buildah

D. skopeo

2. Какие из утилит можно использовать для создания образа из Dockerfile (укажите все правильные ответы)?

A. podman

B. docker

C. buildah

D. skopeo

ЛАБОРАТОРНАЯ РАБОТА 5. КЛАСТЕР KUBERNETES

1. Цель и содержание

Цель лабораторной работы: познокомиться с технологией оркестрации контейнеров.

Задачи лабораторной работы:

– научиться использовать Kubernetes;

– научиться создавать кластеры.

2. Теоретическое обоснование

Kubernetes (от греч. κυβερνήτης, что в переводе означает кормчий) по определению с сайта проекта является «программным обеспечением для автоматического внедрения, масштабирования и управления контейнеризированными приложениями». Также вы можете встретиться с сокращенным написанием названия – K8S. Отличительной особенностью Kubernetes является то, что в основе разработки положен более чем пятнадцатилетний опыт компании Google. В настоящее время разработка управляется специально созданным фондом Cloud Native Computing Foundation (CNCF).

Kubernetes является основой крупнейших облачных инфраструктур. Дадим определение облачным технологиям и месту Kubernetes в них. Устоявшимся в индустрии определением является определение, данное National Institute of Standards and Technology (NIST):

Облачные вычисления – это модель предоставления широкодоступного, удобного доступа по сети к общему пулу настраиваемых вычислительных ресурсов по требованию (к таким как сети, серверы, системы хранения данных, приложения и сервисы). Эти ресурсы

оперативно выделяются и освобождаются при минимальных усилиях, затрачиваемых заказчиком на организацию управления и на взаимодействие с поставщиком услуг.

Этой модели присущи пять основных характеристик, три сервисные модели и четыре модели внедрения. В число характеристик входят: самообслуживание, универсальный доступ по сети, общий пул ресурсов, эластичность и учет потребления.

Сервисные модели различаются по границе контроля потребителем услуг предоставляемой инфраструктуры и включают в себя:

1 инфраструктуру как сервис (IaaS). В данном случае пользователь получает контроль за всеми уровнями стека программного обеспечения, лежащими выше облачной платформы, а именно: виртуальными машинами, сетями, выделенным пользователю объемом пространства на системе хранения данных (СХД). Тогда пользователь выступает администратором операционной системы и всего, что работает поверх, вплоть до приложений. В качестве примеров платформ, обеспечивающих подобную модель, можно назвать OpenStack, Apache CloudStack, Eucalyptus и OpenNebula;

2. программное обеспечение как сервис (SaaS) – в этом случае граница контроля пользователя – само приложение. Пользователь может даже не знать, что такое виртуальная машина или операционная система, он просто работает с приложением. Примеры таких облачных продуктов: Google Docs, Office 365 или, например, Яндекспочта;

3. платформу как сервис (PaaS) – облако, построенное по такой модели, вполне может располагаться «внутри» облака модели IaaS. В этом случае граница контроля пользователя лежит на уровне платформы построения приложений, например сервера приложения, библиотек, среды разработки или базы данных. Пользователь не контролирует и не администрирует виртуальные машины и операционные системы, установленные на них, СХД

и сети. Kubernetes является основой крупнейших облачных инфраструктур типа PaaS.

Четыре модели внедрения облачной платформы включают в себя:

1. Частное облако – вся инфраструктура развернута в центре обработки данных (ЦОД) и служит подразделением одной компании или группы компаний;

2. Публичное облако – заказчиком облачных услуг может выступать любая компания или даже частное лицо. Это модель внедрения, на которой зарабатывают провайдеры облачных услуг;

3. Облако сообщества, или общественное облако. Модель, при которой потребителем является сообщество потребителей из организаций, имеющих общие задачи (например, миссии, требований безопасности, политики и соответствия различным требованиям);

4. Гибридное облако – это комбинация из двух или трех вышеописанных облаков, где разная нагрузка может располагаться, как в частном, публичном или общественном облаке. Как правило, гибридное облако – это больше, чем просто сумма облаков, поскольку ему требуются механизмы и инструменты централизованного управления, распределения и миграции нагрузки между облачными инфраструктурами.

В основу архитектуры Kubernetes были положены разработки проекта Borg – проприетарной системы Google для управления различными облачными приложениями, такими как Google Docs и Gmail. Впервые о Borg было рассказано в 2015 году. И в июле 2015 года была выпущена первая версия Kubernetes. В настоящий момент разработчики придерживаются трехмесячного цикла разработки. Код Kubernetes написан на относительно молодом языке программирования Go, который представляет собой некий гибрид из самых распространенных языков: Java, Python и C++. Крупнейшими компаниямиразработчиками в проекте Kubernetes в настоящее время являются Google и Red Hat. Kubernetes не единственное решение для

управления контейнеризированными приложениями (можно также назвать Docker Swarm, Nomad, Rancher), но, видимо, самое успешное и распространенное.

Kubernetes меняет традиционный подход в написании и развертывании приложений с монолитной на микросервисную архитектуру. При таком подходе приложение состоит из отдельных относительно небольших и самостоятельно разрабатываемых частей, возможно, даже написанных на разных языках программирования. У каждого компонента может быть свой жизненный цикл разработки и обновлений. Компоненты при этом взаимодействуют друг с другом посредством концепции сервисов и вызовов API. Для ознакомления с принципами построения высокомасштабируемых облачных приложений, которые идеально ложатся на инфраструктуру Kubernetes, вы можете изучить принципы «двенадцати факторов».

Рассмотрим архитектуру и компоненты Kubernetes, представленные на рис. 5.1. Это упрощенная диаграмма без учета реализации высокой доступности компонентов. Кластер состоит из как минимум одного управляющего узла и от одного и более вычислительных (или рабочих) узлов, на которых непосредственно запускаются контейнеры. Управляющий узел может быть совмещен с вычислительным, и в самом простом случае все компоненты могут работать в одной виртуальной машине или на одном-единственном сервере. В качестве примера можно привести инструмент Minikube, который позволяет создать такую виртуальную машину в VirtualBox. Возможны варианты также с несколькими управляющими узлами для обеспечения высокой доступности и с единой базой etcd, а также с несколькими управляющими узлами и распределенным кластером etcd.

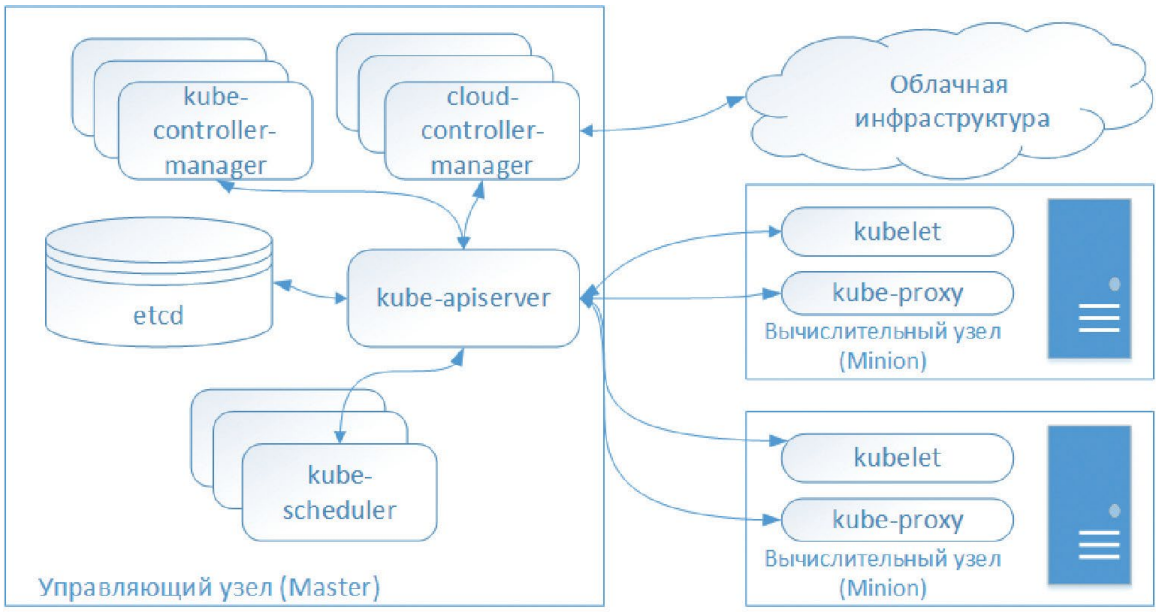

Рисунок 5.1 – Упрощенная архитектура Kubernetes

Рассмотрим сервисы управляющего узла.

kube-apiserver – это центральный компонент кластера Kubernetes. Он является связывающим компонентом для всех остальных сервисов. Все взаимодействие как самих компонентов, так и обращение извне к кластеру проходит через kube-apiserver и валидируется им. Это единственный компонент кластера, который общается с базой данных etcd, где хранится состояние кластера.

etcd – распределенное хранилище типа «ключзначение». Не является «базой данных» в классическом понимании СУБД. Изначально развивалось как составная часть проекта CoreOS. Кластер Kubernetes хранит в etcd всю информацию о состоянии кластера, сервисах, сети и т. д. Доступ к данным осуществляется через REST API. При изменениях записей вместо поиска и изменения старой копии она помечается как устаревшая, а новые значения дописываются в конец. Позже старые значения удаляются специальным процессом.

kube-scheduler – компонент, отвечающий за выбор вычислительного узла, на котором будут запускаться контейнеры (на самом деле podмодули, но мы рассмотрим, что это такое, позже). Существуют механизмы, которые

позволяют вмешаться в этот алгоритм и, например, привязать контейнеры к конкретному узлу. Также на запуск и размещение контейнеров влияют существующие квоты на ресурсы.

kube-controller-manager – компонент, отвечающий за запуск так называемых контроллеров, которые определяют текущее состояние системы. Далее мы рассмотрим примеры контроллеров, таких как, например, Deployment. Фактически kube-controller-manager следит, чтобы кластер и его ресурсы соответствовали заданному состоянию.

cloud-controller-manager – начиная с версии Kubernetes 1.6 этот компонент отвечает за взаимодействие с облачными провайдерами и абстрагирует специфический для конкретного облака код. Ранее за этот функционал отвечал kube-controller-manager.

Перейдем к сервисам вычислительных (управляемых) узлов.

kubelet – данный сервис управляет podмодулями, основываясь на их спецификации. Сервис взаимодействует с kube-apiserver. Это главный сервис для вычислительных узлов.

kubeproxy – отвечает за сетевое взаимодействие podмодулей, управляя правилами iptables (таблица nat).

Среда исполнения контейнеров – в нашем демонстрационном стенде мы будем использовать Docker.

3. Методика и порядок выполнения работы

1. Установите локальный кластер.
2. Установите управляющий узел кластера.
3. Установите рабочие узлы.
4. Установите веб-консоль Kubernetes.

Можно установиь кластер Kubernetes в публичном облаке Microsoft Azure.

4. Вопросы для защиты работы

1. Какой из перечисленных компонентов Kubernetes опциональный?

A. kube-scheduler

B. etcd

C. cloud-controller-manager

D. kube-scheduler

2. Где хранятся настройки доступа к кластеру по умолчанию?

A. k8s/conf

B. kube/configuration

C. kube/config

ЛАБОРАТОРНАЯ РАБОТА 6. ОСНОВЫ РАБОТЫ С KUBERNETES

1. Цель и содержание

Цель лабораторной работы: изучить принципы функционирования и основные объекты Kubernetes.

Задачи лабораторной работы:

– изучить принципы работы с Kubernets с использованием командной строки;

– изучить принципы работы с Kubernets с использованием YAML и JSON.

2. Теоретическое обоснование

Основные термины:

Пространство имен (namespace, ns) – область видимости для ресурсов и объектов в кластере. Не следует путать пространство имен Kubernetes с пространством имен GNU/Linux. Некоторые ресурсы являются ресурсами уровня кластера, и к ним пространства имен не применимы. Примеры таких ресурсов: Node, PersistentVolume или ClusterRole. Можно рассматривать пространство имен как аналог тенанта или проекта в OpenStack.

Pod-модуль (pod, po)– это группа, состоящая из одного или более контейнеров, которые используют общее пространство имен GNU/Linux UTS, Network и IPC и тома. В последних версиях они также могут использовать общее пространство PID. Podмодуль можно рассматривать как «логический сервер». Это базовый тип нагрузки в Kubernetes. Все контейнеры модуля запускаются на одном узле. Все podмодули объединяются одной плоской сетью без NAT.

Метка (label) – пара ключ/значение, которая может быть присвоена любому ресурсу. Селектор (selector) использует метки для фильтрации ресурсов и других операций.

Аннотации (annotations) – позволяют ассоциировать с ресурсами произвольные метаданные, менять свойства или поведение объектов Kubernetes, а также настраивать их взаимодействие с другими сервисами Kubernetes. Аннотации, например, используются в OpenShift для расширения функционала Kubernetes.

Контроллер (controller) – это процесс в Kubernetes, который следит за состоянием ресурсов и, в случае если состояние отличается от заданного, приводит ресурсы к заданному.

Контроллер репликации (replication controller, rc) – базовый тип контроллера. Определяет, как podмодули реплицируются по узлам кластера.

Набор реплик (replica set, rs) – более современная реализация контроллера, имеющего ту же задачу, что и контроллер репликации. Основное отличие – более гибкие правила селекторов. Далее мы познакомимся и с другими типами контроллеров.

Сервис (service, svc) – комбинация IPадреса и порта, предоставляющая доступ к набору podмодулей. По умолчанию сервис подключает клиентов к podмодулям по очереди (поанглийски данный термин звучит round-robin).

Словарь конфигурации ConfigMap (ConfigMap, cm) – набор ключей и значений, который может использоваться другими ресурсами. Как правило, ConfigMap используется для хранения и передачи параметров конфигурации.

Секрет (secret) – аналогичен ConfigMap, но содержит закодированные значения при помощи алгоритма Base64. Секреты по умолчанию не шифруются в etcd, но есть возможность настроить шифрование при установке кластера.

Постоянный том (persistent volume, pv) – постоянный том (хранилище данных), который можно подключить к podмодулю посредством **запроса на постоянный том (persistent volume claim, pvc).**

3. Методика и порядок выполнения работы

Далее мы рассмотрим и некоторые другие ресурсы. Получить список доступных ресурсов и их сокращения можно при помощи команды:

```
[user@master ~]$ kubectl api-resources
```

Прежде чем создать наш первый podмодуль, создадим выделенное для него пространство имен Kubernetes (не путать с пространством имен GNU/Linux!):

```
[user@master ~]$ kubectl create ns test1
Просмотрим список всех пространств имен:
[user@master ~]$ kubectl get ns
NAME STATUS AGE
default Active 78d
kube-public Active 78d kube-system Active 78d
test1 Active 22s
```

Как видно, к трем созданным по умолчанию пространствам имен прибавилось наше новое test1. Перечислим пространства имен, которые уже были созданы во время установки кластера:

- **default** – если явно не задано иное, все объекты по умолчанию создаются в этом пространстве имен;

- **kube-public** – пространство имен, доступное всем без аутентификации. Общая информация о кластере обычно располагается тут;

- **kube-system** – пространство имен, предназначенное для инфраструктурных podмодулей.

Как и для всех ресурсов Kubernetes, определение пространства имен можно сохранить в JSON- или YAML-файл:

```
[user@master ~]$ kubectl get ns test1 -o yaml
apiVersion: v1 kind: Namespace metadata:
creationTimestamp: "2019-03-25T19:32:54Z" name: test1
resourceVersion: "45208" selfLink: /api/v1/namespaces/test1
uid: c92d7436-4f34-11e9-9a7a-0800279bf286 spec:
finalizers:
– kubernetes status:
phase: Active
```

Создадим второе пространство имен через определение из файла yaml. В нем нам не нужна информация времени выполнения. Определение ресурса будет выглядеть так:

```
[user@master ~]$ cat test2.yaml
apiVersion: v1 kind: Namespace metadata:
name: test2
```

Создаем пространство имен и проверяем его наличие:

```
[user@master ~]$ kubectl create -f test2.yaml namespace/test2 created
[user@master ~]$ kubectl get ns
NAME STATUS AGE
default Active 78d
kube-public Active 78d kube-system Active 78d
test1 Active 12m
test2 Active 6s
```

Как мы уже сказали, по умолчанию объект создается в пространстве имен default. Для указания имени пространства имен в команде kubectl можно использовать опцию n с требуемым именем или --all-namespaces для выбора ресурсов из всех пространств имен.

Создадим podмодуль при помощи команды kubectl run, указав в качестве имени модуля и имени образа nginx:

```
[user@master ~]$ kubectl run nginx --image=nginx --port=80 -n test1 --generator=run-pod/v1
```

```
pod/ nginx created
```

Также мы использовали опцию port, указав порт, который должен быть доступен для модуля, пространство имен test1 и опцию --generator=run-pod/v1, которая необходима для того, чтобы был создан только podмодуль без какихлибо контроллеров, которые нам пока не нужны. Проверим, что модуль стартовал:

```
[user@master ~]$ kubectl get pod -n test1 NAME READY STATUS RESTARTS AGE
nginx 1/1 Running 0 5s
```

4. Вопросы для защиты работы

1. Для каких целей служит пространство имен?

A. Ограничение видимости объектов

B. Фильтрация сетевого трафика

C. Создание каталога пользователей

D. Разграничение процессов GNU/Linux

2. Какие объекты может создавать внедрение (укажите все правильные варианты)?

A. Набор реплик

B. Контроллер репликации

C. podмодуль

D. Пространство имен

СПИСОК ЛИТЕРАТУРЫ

1. Вилле К. Представляем C# / Вилле К. – М.: ДМК Пресс, 2008. – 183 с. Режим доступа: http://e.lanbook.com.

2. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. – М.: ДМК Пресс, 2007. – 368 с. Режим доступа: http://e.lanbook.com.

3. Рандольф, Н. Visual Studio 2010 для профессионалов / Н. Рандольф, Д. Гарднер. – М.: Диалектика, 2011. – 1184 с. – ISBN 978-5-8459-1683-9.

4. Уотсон, К. Visual C# 2010. Полный курс / К. Уотсон, К. Нагел. – М.: Вильямс, 2010. – 960 с. – ISBN 978-5-8459-1699-0, 978-0-470-50226-6.

5. Учебное пособие «Технологии программирования I» для студентов специальности 09.03.02 «Информационные системы и технологии» / сост. Николаев Е.И.; рец. Мочалов В.П, Маликов А.В. – Ставрополь : СКФУ, 2011. – 150 с.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по организации и проведению самостоятельной работы
по дисциплине
«Технологии контейнеризации»
для студентов направления подготовки
09.03.02 «Информационные системы и технологии»
направленность (профиль)
«Прикладное программирование в интеллектуальных информационных системах»

Ставрополь, 2026

1. Общие положения

Самостоятельная работа – планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

− формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);

− написание докладов;

− подготовка к семинарам, практическим и лабораторным работам, их оформление;

− составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);

− выполнение учебно-исследовательских работ, проектная деятельность;

− подготовка практических разработок и рекомендаций по решению проблемной ситуации;

− выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;

− компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;

− выполнение курсовых работ (проектов) в рамках дисциплин;

− выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Самостоятельная работа по дисциплине «Технологии контейнеризации» направлена на формирование следующих **компетенций**:

Код, формулировка компетенции	Код, формулировка индикатора
ПК-1 Способен разрабатывать программное обеспечение интеллектуальных информационных систем	ПК-$1_{ИД-1.1}$ – Демонстрирует знания в области применения алгоритмов и их адаптации к практическим задачам
	ПК-$1_{ИД-1.2}$ – Разрабатывает приложения на языке высокого уровня в том числе с использованием интеллектуальных технологий
	ПК-$1_{ИД-1.3}$ – Применяет современные автоматизированные технологии тестирования и отладки программного обеспечения
	ПК-$1_{ИД-1.4}$ – Применяет современные технологии непрерывного развертывания и доставки программного обеспечения
	ПК-$1_{ИД-1.5}$ – Разрабатывает приложения для различных целевых платформ (мобильные, серверные, встраиваемые, оконные)
ПК-7 Способен осуществлять документирование всех стадий жизненного цикла информационных систем, в том числе интеллектуальных	ПК-$7_{ИД-7.1}$ – Демонстрирует знания методик автоматизации составления технической документации и разработки отчетности по утвержденным формам
	ПК-$7_{ИД-7.2}$ – Составляет техническую документацию на всех стадиях реализации проекта с использованием современных инструментальных средств
	ПК-$7_{ИД-7.3}$ – Применяет методы визуализации данных для обоснования эффективности проектов

2. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование набора общенаучных, профессиональных и специальных компетенций будущего бакалавра по соответствующему направлению подготовки

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании

курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

3. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателем	Всего
8 семестр					
ПК-1	Самостоятельное изучение литературы и источников	Собеседование	2	2	4
ПК-7	Подготовка к лабораторным занятиям	Защита ЛР	48	16	64
Итого за 8 семестр			50	18	68
Итого			50	18	68

4. Порядок выполнения самостоятельной работы студентом

4.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях)

дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют ***четыре основные установки в чтении научного текста***:

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;
2. Выделите главное, составьте план;
3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;
4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.

5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

4.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы практические и лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

4.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует

помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

4.4. Методические рекомендации по написанию научных текстов (докладов, докладовов, эссе, научных статей и т.д.)

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста - это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время – важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремясь структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важным мест и т, п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору (исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Доклад - это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Доклад не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в доклад е должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки

а студентом.

Выполнение доклада начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- обсуждение цели и задач работы, основных моментов избранной темы;
- консультирование по вопросам подбора литературы;
- составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы - изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания доклада. Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста доклада предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки доклад сдается на кафедру для его оценивания руководителем.

Требования к написанию доклада

Написание 1 доклада является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема доклада может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласоваться с преподавателем.

Доклад должен быть написан научным языком.

Объем доклада должен составлять 20-25 стр.

Структура доклада:

- Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.
- Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению. Изложение материала не должно ограничиваться лишь описательным подходом к раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.
- Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса
- Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.
- Приложение (при необходимости).

Требования к оформлению:

- текст с одной стороны листа;

- шрифт Times New Roman;
- кегль шрифта 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
- доклад должен быть представлен в сброшюрованном виде.

Порядок защиты доклада:

Защита доклада проводится на практических занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту доклада отводится 5-7 минут времени, в ходе которого студент должен показать свободное владение материалом по заявленной теме. При защите доклада приветствуется использование мультимедиа-презентации.

Оценка доклада

Доклад оценивается по следующим критериям:

• соблюдение требований к его оформлению;

• необходимость и достаточность для раскрытия темы приведенной в тексте доклада информации;

• умение студента свободно излагать основные идеи, отраженные в докладе;

• способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

Критерии оценки:

Оценка «отлично» выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

Оценка «хорошо» выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

Оценка «удовлетворительно» выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

Оценка «неудовлетворительно» выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

4.5. Методические рекомендации по выполнению исследовательских проектов

Исследовательская проектная работа – это групповая работа, для выполнения которой необходим выбор и приложение научной методики к поставленной задаче, получение собственного теоретического или экспериментального материала, на основании которого необходимо провести анализ и сделать выводы об исследуемом явлении. Выполнение проекта – это всегда коллективная, творческая практическая работа, предназначенная для получения определенного продукта или научно-технического результата. Такая работа подразумевает четкое, однозначное формирование поставленной задачи, определение сроков выполнения намеченного, определение требований к разрабатываемому объекту.

Выполнение 1 группового проекта является обязательным условием выполнения самостоятельной работы по любой дисциплине профессионального цикла. Тема

проектного задания может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласоваться с преподавателем.

Требования по выполнению и оформлению проекта

При выполнении проекта приветствуется работа в группе (2-3 человека). Проект – это исследовательская работа, в ходе которой студенты должны продемонстрировать владение навыками научного исследования, умения проводить анализ, обобщать информацию, делать выводы, предлагать свои решения проблемы, рассматриваемой в проекте.

При подготовке материалов проекта студенты должны продемонстрировать владение современными методами компьютерной обработки данных.

Критерии оценки работы участника проекта.

Для каждого из участников проекта оцениваются:

• профессиональные теоретические знания в соответствующей области;

• умение работать со справочной и научной литературой, осуществлять поиск необходимой информации в Интернет;

• умение работать с техническими средствами;

• умение пользоваться соответствующими выполняемому проекту информационными технологиями;

• умение готовить материалы проекта для презентации: составлять и редактировать тексты, формировать презентацию проекта;

• умение работать в команде;

• умение публично представлять результаты собственной деятельности;

• коммуникабельность, инициативность, творческие способности.

Критерии выставления оценки участникам проекта

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
«Отлично»	Работа выполнена на высоком профессиональном уровне. Представленный материал в основном фактически верен, допускаются негрубые фактические неточности. Студент	Технические средства и информационные технологии освоены и использованы для реализации проекта полностью	Студент проявил инициативу, творческий подход, способность к выполнению сложных заданий, навыки работы в коллективе, организационны	Проект представлен полностью и в срок.

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
	свободно отвечает на вопросы, связанные с проектом.		е способности.	
«Хорошо»	Работа выполнена на достаточно высоком профессиональном уровне. Допущено до 4–5 фактических ошибок. Студент отвечает на вопросы, связанные с проектом, но недостаточно полно.	Обнаруживаются некоторые ошибки в использовании соответствующих технических средств и информационных технологий	Студент достаточно полно, но без инициативы и творческих находок выполнил возложенные на него задачи.	Проект представлен достаточно полно и в срок, но с некоторыми недоработками.
«Удовлетворительно»	Уровень недостаточно высок. Допущено до 8 фактических ошибок. Студент может ответить лишь на некоторые из заданных вопросов, связанных с проектом.	Обнаруживает недостаточное владение навыками работы с техническими средствами и соответствующими информационными технологиями	Студент выполнил большую часть возложенной на него работы.	Проект сдан со значительным опозданием (более недели) и не полностью
«Неудовлетворительно»	Работа не выполнена или выполнена на низком уровне. Допущено более 8 фактических ошибок. Ответы на связанные с проектом вопросы обнаруживают непонимание предмета и отсутствие ориентации в материале проекта.	Навыков работы с техническими средствами нет, информационные технологии не освоены	Студент практически не работал, не выполнил свои задачи или выполнил лишь отдельные не существенные поручения в групповом проекте.	Проект не сдан.

Студенты должны: защитить проект в режиме презентации, предъявить файлы выполненного проекта, уметь рассказать о технологиях, использованных ими при выполнении проекта, дать оценку работы каждого члена группы (*если проект групповой*).

4.6. Методические рекомендации по подготовке к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению практических задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неутомительные занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Контроль самостоятельной работы студентов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование, оценка доклада, оценка презентации, оценка участия в круглом столе, оценка выполнения проекта.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

Список литературы для выполнения СРС

Перечень основной литературы:

1. Зиборов, В. В. Visual C# 2012 на примерах / Виктор Зиборов. - Санкт-Петербург : БХВ-Петербург, 2013. - 473 с. : ил. ; 24 см. - ISBN 978-5-9775-0888-9.

2. Баженова, И. Ю. Основы проектирования приложений баз данных / И. Ю. Баженова. – 2-е изд., испр. – М.: Национальный Открытый Университет «ИНТУИТ», 2016. – 238 с. [Электронный ресурс]. Режим доступа: http://biblioclub.ru/index.php?page=book&id=428933"id=428933

Перечень дополнительной литературы:

1. Основы высокопроизводительных вычислений : учебное пособие / К.Е. Афанасьев, С.В. Стуколов, В.В. Малышенко, С.Н. Карабцев, Н.Е. Андреев. - Кемерово : Кемеровский государственный университет, 2012. - 412 с. - http://biblioclub.ru/. - ISBN 978-5-8353-1246-7

2. Уильямс, Э. Параллельное программирование на C++ в действии : Практика разработки многопоточных программ / Энтони Уильямс ; [пер. с англ. Слинкина А. А.]. - М. : ДМК Пресс, 2012. - 672 с. : ил. - Прил.: с. 437-653. - Предм. указ.: с. 654-671. - ISBN 978-5-9407-448-1

Методическая литература:

1. Методические рекомендации для самостоятельной работы студентов по дисциплине «Технологии контейнеризации»
2. Методические указания к лабораторным работам по дисциплине «Технологии контейнеризации»

Интернет-ресурсы:

1. http://el.ncfu.ru/ – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Технологии контейнеризации».
2. http://metanit.com - сайт посвящен различным языкам и технологиям программирования, компьютерам, мобильным платформам и ИТ-технологиям
3. https://professorweb.ru - информационный ресурс, посвященный разработке приложений на основе технологии .NET.