### ****Тема 1: Введение в DevOPS****

#### ****Урок 1: Что такое DevOPS и основные принципы****

**Теория:**

DevOPS — это методология, направленная на улучшение взаимодействия между разработчиками и операционными командами.

Основные цели DevOPS:

Ускорение разработки и выпуска продуктов;

Повышение качества и стабильности приложений;

Автоматизация процессов.

Основные принципы:

**Культура сотрудничества** — между разработкой и операционными командами;

**Автоматизация процессов** — CI/CD, тестирование, деплой;

**Непрерывное улучшение** — постоянное улучшение процессов и обратная связь.

Задачи DevOPS:

Автоматизация сборки, тестирования, деплоя;

Управление инфраструктурой как код (Infrastructure as Code);

Мониторинг и логирование.

**Тест:**

1. Какая главная цель DevOPS?  
   → **Правильный ответ:** ускорение разработки и выпуска продуктов
2. Что означает принцип "Инфраструктура как код"?  
   → **Правильный ответ:** Управление инфраструктурой с помощью кода
3. Какую задачу решает DevOPS, помимо автоматизации?  
   → **Правильный ответ:** Сотрудничество между разработчиками и операционными командами

#### ****Урок 2: Инструменты DevOPS****

**Теория:**

DevOPS использует различные инструменты для автоматизации, управления и мониторинга. Некоторые из них:

**CI/CD**: Jenkins, GitLab CI, CircleCI.

**Контейнеризация**: Docker, Kubernetes.

**Управление конфигурациями**: Ansible, Puppet, Chef.

**Управление инфраструктурой**: Terraform, Packer.

**Мониторинг и логирование**: Prometheus, Grafana, ELK Stack.

**Тест:**

1. Какой инструмент используется для автоматизации сборки и деплоя в DevOPS? Ответом является одно понятие  
   → **Правильный ответ:** jenkins
2. Для чего используется Kubernetes в DevOPS? В качестве ответа введите одно слово вместо многоточия: Для …  
   → **Правильный ответ:** оркестрации
3. Какой инструмент используется для управления конфигурацией?  
   Варианты ответов:
   1. Docker
   2. Kubernetes
   3. Ansible
   4. Jenkins
   5. Grafana

→ **Правильный ответ:** Ansible

### ****Тема 2: Модель управления инфраструктурой Packer, Terraform****

#### ****Урок 1: Введение в Packer и создание образов****

**Теория:**

Packer — это инструмент для автоматизации создания виртуальных машин, контейнеров и других образов. Он позволяет создать несколько образов из единого исходного кода.

Основные концепции:

**Конфигурация шаблона**: описание, как создать образ;

**Провайдеры**: поддерживаемые платформы, такие как AWS, VirtualBox, VMware;

**Пост-обработчики**: действия после создания образа.

Основные команды:

packer init — инициализация рабочей среды;

packer validate — проверка конфигурации;

packer build — создание образа.

**Тест:**

1. Что делает команда packer build?  
   → **Правильный ответ:** Создает образ
2. Какой инструмент используется для создания виртуальных машин и контейнеров?  
   → **Правильный ответ:** Packer
3. Что такое провайдер в Packer?  
   → **Правильный ответ:** Платформа для создания образов

#### ****Урок 2: Введение в Terraform и управление инфраструктурой как код****

**Теория:**

Terraform — это инструмент для управления инфраструктурой как код (IaC). Он позволяет описывать инфраструктуру в виде кода и управлять ею через версии, что упрощает автоматизацию и воспроизводимость.

Основные понятия:

**Ресурсы** — элементы инфраструктуры, такие как серверы, сети, базы данных;

**Модули** — повторно используемые блоки кода;

**Состояние (State)** — хранение информации о текущем состоянии инфраструктуры.

Основные команды:

terraform init — инициализация проекта;

terraform plan — планирование изменений;

terraform apply — применение изменений;

terraform destroy — уничтожение инфраструктуры.

**Тест:**

1. Какой инструмент используется для управления инфраструктурой как код?  
   → **Правильный ответ:** Terraform
2. Какая команда используется для применения изменений в инфраструктуре?  
   → **Правильный ответ:** terraform apply
3. Что такое модуль в Terraform?  
   → **Правильный ответ:** Повторно используемый блок кода

### ****Тема 3: Управление конфигурацией Ansible****

#### ****Урок 1: Введение в Ansible и его архитектура****

**Теория:**

Ansible — это инструмент для автоматизации управления конфигурациями. Ansible использует концепцию **Push-модели**, когда изменения настраиваются на удаленных машинах.

Основные компоненты:

**Playbook** — сценарии, которые описывают, как управлять системами;

**Inventory** — файл, содержащий список управляемых узлов;

**Module** — функции, которые выполняются на удаленных системах.

Команды:

ansible all -m ping — проверка доступности всех узлов;

ansible-playbook playbook.yml — выполнение playbook;

ansible -i hosts all -m yum -a "name=httpd state=installed" — установка пакета через Ansible.

**Тест:**

1. Что такое Playbook в Ansible?  
   → **Правильный ответ:** сценарий для управления конфигурацией
2. Какая модель используется в Ansible для управления конфигурациями?  
   → **Правильный ответ:** Push
3. Как выполнить playbook в Ansible?  
   → **Правильный ответ:** ansible-playbook playbook.yml

#### ****Урок 2: Пример Playbook и использование переменных****

**Теория:**

Playbook в Ansible — это последовательность задач, которые описывают, как настроить систему. Также можно использовать **переменные** для динамической настройки.

Пример playbook:

- hosts: all

become: yes

tasks:

- name: Установить nginx

yum:

name: nginx

state: present

- name: Запустить nginx

service:

name: nginx

state: started

Переменные могут быть:

Локальные, определенные в playbook;

Глобальные, определенные в инвентаре или в отдельном файле.

**Тест:**

1. Какой модуль используется для установки пакетов в Ansible?  
   → **Правильный ответ:** yum
2. Как задать переменную в Ansible?  
   → **Правильный ответ:** Через vars в Playbook
3. Какой параметр в playbook выполняет задачу с правами суперпользователя?  
   → **Правильный ответ:** become: yes

### ****Тема 4: Работа с Docker****

#### ****Урок 1: Введение в Docker и создание контейнеров****

**Теория:**

Docker — это платформа для контейнеризации приложений. Контейнеры изолируют приложение и его зависимости от основной операционной системы.

Основные команды:

docker run hello-world — проверить, установлен ли Docker;

docker build -t myapp . — сборка контейнера из Dockerfile;

docker run -d -p 8080:80 myapp — запуск контейнера в фоновом режиме;

docker ps — показать запущенные контейнеры;

docker stop <container\_id> — остановить контейнер.

**Тест:**

1. Как запустить контейнер из образа?  
   → **Правильный ответ:** docker run
2. Как проверить, какие контейнеры запущены?  
   → **Правильный ответ:** docker ps

#### ****Урок 2: Работа с Docker Compose****

**Теория:**

Docker Compose — это инструмент для определения и запуска многоконтейнерных Docker приложений. Он использует файл docker-compose.yml для конфигурации.

Пример файла docker-compose.yml:

version: '3'

services:

web:

image: nginx

ports:

- "8080:80"

db:

image: postgres

environment:

POSTGRES\_PASSWORD: example

Основные команды:

docker-compose up — запускает все контейнеры из docker-compose.yml;

docker-compose down — останавливает и удаляет контейнеры.

**Тест:**

1. Что такое Docker Compose?  
   → **Правильный ответ:** Инструмент для работы с многоконтейнерными приложениями
2. Какой командой можно запустить контейнеры из файла docker-compose.yml?  
   → **Правильный ответ:** docker-compose up
3. Какая команда останавливает и удаляет контейнеры в Docker Compose?  
   → **Правильный ответ:** docker-compose down

### ****Тема 5: Jenkins****

#### ****Урок 1: Введение в Jenkins и настройка первого проекта****

**Теория:**

Jenkins — это инструмент для автоматизации различных задач, таких как сборка, тестирование и деплой приложений. Он широко используется для реализации процессов **CI/CD (Continuous Integration/Continuous Deployment)**.

Основные компоненты Jenkins:

**Jenkins Pipeline** — сценарии для автоматизации рабочих процессов;

**Jenkins Jobs** — задачи, которые Jenkins выполняет, например, сборка кода;

**Jenkins Master** — главный сервер Jenkins;

**Jenkins Agents** — дополнительные серверы для выполнения задач.

Процесс создания первого проекта:

1. Установите Jenkins на сервере.
2. Создайте новый проект (Job).
3. Настройте сборку проекта с помощью инструмента Git.
4. Настройте выполнение тестов и деплой.

**Тест:**

1. Что делает Jenkins?  
   → **Правильный ответ:** автоматизирует процессы сборки, тестирования и деплоя
2. Что такое Jenkins Pipeline?  
   → **Правильный ответ:** сценарий для автоматизации процессов
3. Что такое Jenkins Agent?  
   → **Правильный ответ:** дополнительный сервер для выполнения задач

#### ****Урок 2: Создание Jenkins Pipeline для CI/CD****

**Теория:**

Jenkins Pipeline позволяет создавать автоматизированные процессы для CI/CD. Существует два типа Pipeline:

**Declarative Pipeline** — декларативный синтаксис для описания пайплайна;

**Scripted Pipeline** — более гибкий и расширяемый скриптовый синтаксис.

Пример простого декларативного Pipeline:

pipeline {

agent any

stages {

stage('Build') {

steps {

script {

sh 'mvn clean install'

}

}

}

stage('Test') {

steps {

script {

sh 'mvn test'

}

}

}

stage('Deploy') {

steps {

script {

sh 'scp target/myapp.war user@server:/path/to/deploy'

}

}

}

}

}

Важные этапы:

**Build** — сборка приложения;

**Test** — выполнение тестов;

**Deploy** — деплой приложения.

**Тест:**

1. Какой синтаксис используется для создания Jenkins Pipeline?  
   → **Правильный ответ:** Groovy
2. Какой этап Pipeline выполняет сборку приложения?  
   → **Правильный ответ:** Build
3. Какой командой можно выполнить деплой приложения?  
   → **Правильный ответ:** scp

### ****Тема 6: Kubernetes****

#### ****Урок 1: Введение в Kubernetes и его архитектура****

**Теория:**

Kubernetes — это система для оркестрации контейнеров, предназначенная для автоматизации развертывания, масштабирования и управления контейнеризованными приложениями.

Основные компоненты Kubernetes:

**Pod** — минимальная единица развертывания, содержащая контейнеры;

**Node** — физический или виртуальный сервер, на котором запускаются Pods;

**Cluster** — набор Node, управляющийся Kubernetes;

**Control Plane** — управляющая плоскость, которая принимает решения о состоянии кластера (например, Kubernetes API Server);

**Kubernetes Services** — абстракция, которая позволяет доступ к контейнерам.

Kubernetes использует файлы манифестов YAML для описания развертываемых объектов, таких как Pods, Deployments, Services.

**Тест:**

1. Что такое Pod в Kubernetes?  
   → **Правильный ответ:** минимальная единица развертывания, содержащая контейнеры
2. Какой компонент в Kubernetes управляет состоянием кластера?  
   → **Правильный ответ:** control plane
3. Какой формат используется для описания объектов Kubernetes?  
   → **Правильный ответ:** yaml

#### ****Урок 2: Управление приложениями в Kubernetes****

**Теория:**

Kubernetes позволяет управлять приложениями с помощью объектов, таких как:

**Deployment** — описывает развертывание приложения и его обновления;

**Service** — создает стабильный доступ к подам;

**Ingress** — управляет внешним доступом к приложениям в кластере.

Основные команды Kubernetes:

kubectl get pods — список всех подов;

kubectl create -f deployment.yml — создание объекта из манифеста;

kubectl apply -f deployment.yml — применение изменений;

kubectl delete -f deployment.yml — удаление объекта.

Пример Deployment манифеста:

apiVersion: apps/v1

kind: Deployment

metadata:

name: myapp

spec:

replicas: 3

selector:

matchLabels:

app: myapp

template:

metadata:

labels:

app: myapp

spec:

containers:

- name: myapp

image: myapp:latest

ports:

- containerPort: 8080

**Тест:**

1. Какой объект Kubernetes описывает развертывание приложения?  
   → **Правильный ответ:** deployment
2. Как создать объект в Kubernetes из файла манифеста?  
   → **Правильный ответ:** kubectl create -f
3. Как удалить объект в Kubernetes?  
   → **Правильный ответ:** kubectl delete -f

### ****Тема 7:**** Мониторинг (Prometheus & Grafana)

#### ****Урок 1: Введение в мониторинг с Prometheus****

**Теория:**

Prometheus — это система мониторинга и сбора метрик, которая широко используется для сбора и хранения временных рядов данных. Prometheus предоставляет язык запросов **PromQL** для выборки и анализа данных.

Основные компоненты:

**Prometheus Server** — основной компонент для сбора и хранения метрик;

**Exporter** — инструмент, собирающий метрики с приложений и сервисов;

**Alertmanager** — система уведомлений и оповещений.

Основные шаги для настройки:

Установите Prometheus.

Настройте **targets** для сбора метрик.

Настройте **Alertmanager** для уведомлений.

**Тест:**

1. Какой язык используется для запросов в Prometheus?  
   → **Правильный ответ:** promql
2. Какой компонент Prometheus собирает метрики с приложений?  
   → **Правильный ответ:** exporter
3. Какую задачу выполняет Alertmanager?  
   → **Правильный ответ:** управление уведомлениями

#### ****Урок 2: Визуализация данных с Grafana****

**Теория:**

Grafana — это инструмент для визуализации данных, который может интегрироваться с Prometheus и другими источниками данных. Он позволяет создавать дашборды с графиками, метками и алертами.

Основные шаги:

Установите Grafana.

Настройте источник данных Prometheus.

Создайте дашборд и добавьте панели с графиками.

Пример панели:

Тип панели: Graph

Источник данных: Prometheus

Запрос: rate(http\_requests\_total[5m])

**Тест:**

1. Какой инструмент используется для визуализации данных в DevOPS?  
   → **Правильный ответ:** grafana
2. Какой источник данных используется в Grafana для интеграции с Prometheus?  
   → **Правильный ответ:** prometheus
3. Что отображает панель типа "Graph" в Grafana?  
   → **Правильный ответ:** графики

### ****Тема 8:** ELK & GRAY LOG**

#### ****Урок 1: Введение в ELK Stack****

**Теория:**

ELK Stack (Elasticsearch, Logstash, Kibana) — это набор инструментов для сбора, анализа и визуализации логов.

Компоненты:

**Elasticsearch** — поисковая и аналитическая система для хранения и обработки логов;

**Logstash** — инструмент для сбора и обработки логов;

**Kibana** — интерфейс для визуализации и анализа данных из Elasticsearch.

Процесс работы ELK Stack:

**Logstash** получает логи и отправляет их в **Elasticsearch**.

**Elasticsearch** хранит и индексирует логи.

**Kibana** предоставляет интерфейс для поиска и визуализации данных.

**Тест:**

1. Какой компонент ELK Stack отвечает за визуализацию данных?  
   → **Правильный ответ:** kibana
2. Какой компонент отвечает за хранение и обработку логов?  
   → **Правильный ответ:** elasticsearch
3. Что собирает и обрабатывает Logstash в ELK Stack?  
   → **Правильный ответ:** логи

#### ****Урок 2: Настройка ELK Stack для логирования****

**Теория:**

Настройка ELK Stack включает установку всех трех компонентов и их настройку для сбора и анализа логов.

Установите **Elasticsearch**, **Logstash** и **Kibana**.

Настройте **Logstash** для сбора логов, например, из файлов или системных журналов.

Настройте **Kibana** для визуализации данных, создавая дашборды для логов.

Пример конфигурации Logstash для получения логов из файлов:

input {

file {

path => "/var/log/syslog"

start\_position => "beginning"

}

}

output {

elasticsearch {

hosts => ["http://localhost:9200"]

}

}

**Тест:**

1. Что необходимо настроить в Logstash для получения логов?  
   → **Правильный ответ:** input
2. Какой компонент ELK Stack отвечает за создание дашбордов?  
   → **Правильный ответ:** kibana
3. Где хранятся логи в ELK Stack?  
   → **Правильный ответ:** elasticsearch