

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА ПРИКЛАДНОЇ МАТЕМАТИКИ ТА ІНФОРМАТИКИ**

ЕЛЕКТРОННИЙ НАВЧАЛЬНО-НАОЧНИЙ ПОСІБНИК

**СХЕМОКОНСПЕКТ ЛЕКЦІЙ
з навчальної дисципліни
«ТЕХНОЛОГІЇ DEVOPS»
для студентів всіх форм навчання
галузі знань 12 Інформаційні технології**

Луцьк, 2023

Схемоконспект лекцій з навчальної дисципліни «Технології DevOPS» для студентів всіх форм навчання галузі знань 12 Інформаційні технології : навчально-наочний посібник / уклад. Я.Ю. Дорогий. – Луцьк : ДонНТУ, 2023. – 766 с.

Навчальна дисципліна «Технології DevOPS» є актуальною і затребуваною в контексті підготовки сучасних фахівців у галузі інформаційних технологій та кібербезпеки та з огляду на формування когнітивних, афективних та моторних компетентностей в контексті вивчення та розуміння принципів організації процесу швидкого розгортання, супроводу, моніторингу та загальної автоматизації певних задач в різних середовищах на Linux-сумісних платформах..

Пропонований студентам для опанування схемоконспект лекцій з цієї дисципліни має за мету забезпечити їм певну допомогу під час вивчення ними ключових положень базових тем курсу. Схематична форма конспекту акцентує увагу студентів на закріпленні основних фахових знань, формуванні відповідних умінь і в цілому отриманні ними кваліфікаційних компетентностей.

Укладач: _____ Я. Ю. Дорогий, доцент, д. т. н., професор кафедри ПМІ

Рецензент: _____ В. В. Цуркан, доцент, к. т. н., доцент спеціальної кафедри №5 ІСЗІ КПІ ім. Ігоря Сікорського,

Відповідальний за випуск: Н. О. Маслова, доцент, к. т. н., зав. каф. ПМІ

Затверджено навчально-методичним відділом ДонНТУ, протокол № 5 від 16.01.2024 р.

Розглянуто на засіданні кафедри прикладної математики і інформатики, протокол № 13 від 27.12.2023 р.

ЗМІСТ

ВСТУП		5
Тема 1.	Вступ до DevOPS.	
	Лекція 1. Вступ до DevOPS.	6
Тема 2.	Основи застосування системи контролю версій Git.	
	Лекція 2. Системи керування версіями.	26
	Лекція 3. Git. Віддалені репозиторії. Розгалуження.	82
Тема 3.	Bash та Python для DevOps.	
	Лекція 4. Основи BASH.	132
	Лекція 5. Автоматизація з Python.	192
Тема 4.	Інструментарії віртуалізації та розгортання.	
	Лекція 6. Основи Vagrant.	224
	Лекція 7. Основи Capistrano.	265
Тема 5.	Система контейнеризації Docker.	
	Лекція 8. Основи Docker.	316

Тема 6.	Системи управління конфігураціями.	
	Лекція 9. Система управління конфігураціями Chef.	352
	Лекція 10. Система управління конфігураціями Puppet.	401
	Лекція 11. Система управління конфігураціями Ansible.	438
Тема 7.	Системи безперервної інтеграції.	
	Лекція 12. Система безперервної інтеграції Jenkins.	531
Тема 8.	Системи оркестрації.	
	Лекція 13. Вступ до Kubernetes.	594
	Лекція 14. Конфігурація Kubernetes.	633
Тема 9.	Системи моніторингу та налаштування сервісів.	
	Лекція 15. Система моніторингу Prometheus.	664
	Лекція 16. Система налаштування сервісів інфраструктури Consul.	715
	СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ ДЛЯ САМОПІДГОТОВКИ	758

У контексті швидко змінюючогося й вимагаючого середовища, де висока мінливість та невизначеність стають необхідністю, функціонування сучасних держав та господарських суб'єктів пов'язане з підвищеним ризиком. Ця реальність викликає появу та активність різноманітних загроз як на зовнішньому, так і на внутрішньому рівнях.

Негативні наслідки цих загроз можуть призвести до серйозних наслідків, тому протидія їм та відновлення після атак є актуальним завданням. Для ефективного вирішення цих завдань необхідно впроваджувати сучасні технології, а однією з ключових складових є методологія та інструментарій DevOps.

Курс «Технології DevOps» спрямований на формування у студентів глибокого розуміння та практичних навичок у сфері розробки та впровадження DevOps-процесів. Основний акцент курсу робиться на автоматизації, забезпеченні безпеки та ефективного управління життєвим циклом програмного забезпечення.

Студентам пропонується активна участь у виконанні практичних завдань та лабораторних робіт, що дозволить їм здобути практичний досвід і вміння використовувати сучасні інструменти для автоматизації та оптимізації розробки.

Курс буде корисним не лише для тих, хто прагне вдосконалити свої знання в області розробки, а й для фахівців, які мають на меті підвищити ефективність та безпеку робочих процесів у галузі програмного забезпечення.

Донецький національний технічний університет



Тема 1. Вступ до DevOps.

Лекція 1. Вступ до дисципліни

Предмет: Технології DevOPS

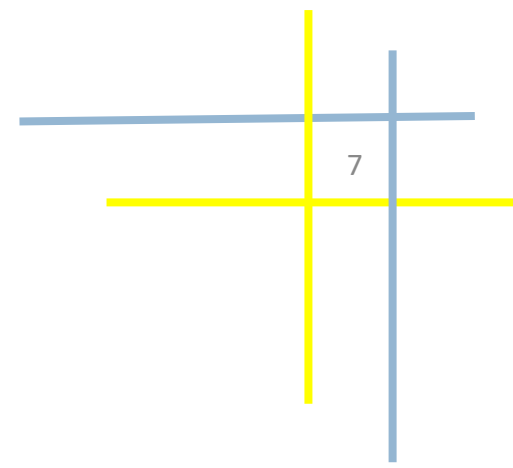
Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання



- ☐ Мета та структура курсу
- ☐ Вступ в DevOps
- ☐ Continuous Integration
- ☐ Середовища
- ☐ Continuous Delivery та Deployment
- ☐ Continuous Testing
- ☐ Метрики



Мета курсу

8

- ❑ Метою дисципліни "Технології DevOps" є формування системи теоретичних та практичних знань та навичок щодо процесу швидкого розгортання, супроводу, моніторингу та загальної автоматизації певних задач в різних середовищах на Linux-сумісних платформах.



Очікувані результати

Внаслідок вивчення курсу студенти повинні вміти:

- ☐ застосовувати отримані навички самостійного вивчення навчальної та наукової літератури, володіти понятійним апаратом;
- ☐ застосовувати систему контролю версій Git;
- ☐ використовувати Bash та Python для автоматизації задач;
- ☐ розгортати, налаштовувати та використовувати інструментарій віртуалізації Vagrant та систему розгортання Capistrano;
- ☐ використовувати систему контейнеризації Docker;
- ☐ розгортати, налаштовувати та використовувати системи управління конфігураціями Chef, Puppet та Ansible;
- ☐ розгортати, налаштовувати та використовувати систему безперервної інтеграції Jenkins;
- ☐ розгортати, налаштовувати та використовувати систему оркестрації Kubernetes;
- ☐ розгортати, налаштовувати та використовувати системи моніторингу та налаштування сервісів.



Структура курсу

- ☐ 16 лекцій
- ☐ 14 лабораторних робіт
- ☐ Екзамен



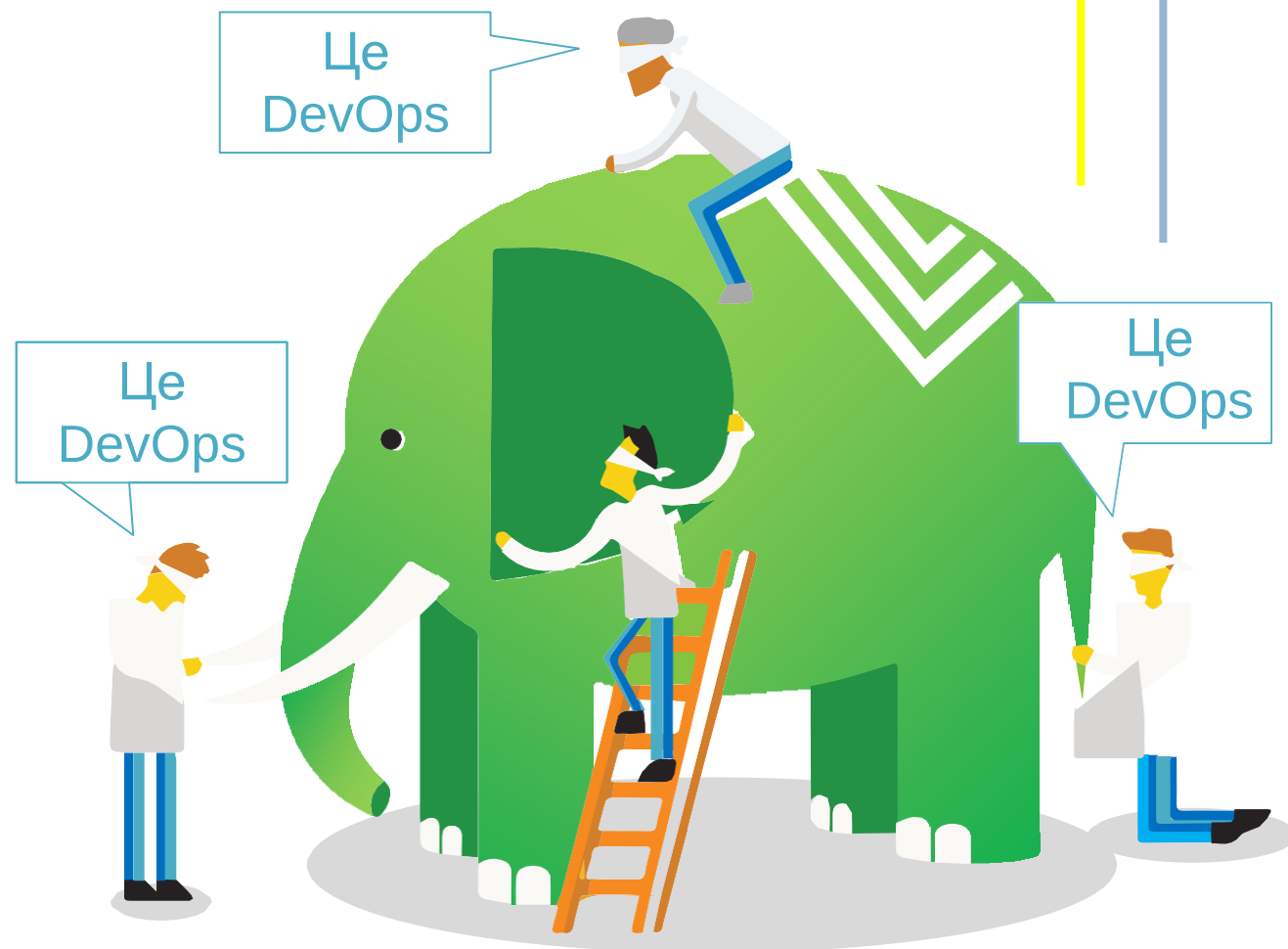
Що таке DevOps

DevOps* – це інженерна культура, практики та інструменти, які направлені на скорочення релізного циклу, підвищення ефективності та забезпечення можливості випуска релізу в будь-який момент шляхом:

- інтеграції IT Development та IT Operations;
- створення єдиної команди;
- автоматизації рутинних операцій;
- частоті перевірки коду

! МЕТА: скоротити T2M* та підвищити гнучкість впровадження (release anytime)
■ без втрати якості та надійності

- DevOps – акронім від англ. development та operations
- T2M (time-to-market) – строк від початку розробки продукту до його виходу на ринок





Упередження щодо DevOps

Хтось з IT: А що, DevOps буде замість Agile?

Хтось з IT: DevOps та Waterfall несумісні!

Менеджер IT: Ми в ці вихідні не працюємо, у вас буде час прийти та встановити мені DevOps.

Dev: DevOps - це задача адміністраторів.

Dev: У мене установка на середовища розробки йде однією кнопкою.
Я зробив DevOps!

Dev: Мені DevOps не потрібен. Ми всі генії розробки, а Вася робить збірку навіть під час сну.

Питання: Що буде, якщо кількість задач збільшиться, в вашу команду додатуться нові розробники, а Вася піде в декрет?

Dev: Це проблема нових розробників, а Вася ...



Ops: Ви розробіть процес розгортання та покажіть, а ми надамо свої зауваження.

Ops: У нас і так все добре та надійно. Не потрібні нам нові інструменти. **Питання:** Що буде, якщо не один реліз в місяць, а два в тиждень?

Ops: Терміново збільшуємо штат!

Ops: Ну добре, впровадили DevOps. Де моя одна кнопка?

Менеджер IT: Потрібно активно попрацювати три місяці, зробимо DevOps т а тоді розслабимося

Питання: Вас не бентежить слово Continuous в назвах практик DevOps?

Ops: Е-е-е ...



Основні компоненти DevOps



Тільки змінив всі чотири складові, можна впровадити DevOps!



DevOps принципи

Culture

- Розвиток інженерної культури – від розробника до інженера
- Від відповідальності за етап– до відповідальності за результат

Automation

- Реліз в будь-який час
- Прийняття змін та зменшення ризиків

Measurement

- Безперервне покращення, яке базується на метриках та реальних даних

Sharing

- Колаборація та відкрита комунікація
- Постійний зворотній зв'язок «справа наліво»



Основні практики DevOps, пов'язані в циклічний автоматизований процес

Безперевне розгортання (Continuous Deployment)

Автоматичне розгортання на середовищах аж до промислового

Безперевна доставка (Continuous Delivery)

Автоматичне розгортання та тестування

Безперервна інтеграція (CI*)

Автоматична збірка, запуск модульних тестів та статичного аналізу

1



Кодування

2



Модульне
тестування

3



Інспекція кода,
статичний
аналіз коду та
ІБ

4



Збірка та
розгортання
на середовищах
розробки

5



Розгортання
на СТ

6



Тестування
на СТ

7



Розгортання
на ІФТ, НТ, ІБ

8



Тестування
на ФТ, НТ, ІБ

9



Реєстрація в
сховище
дистрибутивів
організації

10



Розгортання та
проведення ПВ
/ Hotfix

11



Розгортання
на ПРОД

- Підвищення якості коду за рахунок:
 - о практик модульного тестування (unit testing)
 - о іспекції коду (code review) та статичного аналізу коду на відповідність кращим світовим стандартам кодування та безпеки
- Скорочення строків розробки за рахунок:
 - о прискорення циклу зворотнього зв'язку
 - о збільшення частоти збірок (мінімум 1 раз на день)

- Підвищення якості тестування за рахунок:
 - о впровадження практик автоматизованого тестування - testing automation
 - о впровадження динамічного аналізу застосунків на вразливість
- Скорочення часу виведення в ПРОД за рахунок:
 - о автоматизації розгортання дистрибутивів на середовища
 - о динамічного виділення інфраструктури, infrastructure as code

- Підвищення частоти введення релізів в ПРОД за рахунок:
 - о впровадження практик автоматизації розгортання

*Continuous Integration



Практики автоматизованого тестування Continuous Testing



Види автоматизованого тестування

Що це таке?

Хто розробляє та проводить тести?

Модульне тестування – unit testing	Перевірка працездатності частини коду (функції, класу і т.д.). Тести пишуться на мові програмування з використанням спеціальних фреймворків	Інженер Agile - команди
Статичний аналіз коду та ІБ	Перевірка коду на відповідність міжнародним стандартам мов програмування та написання безпечного коду за допомогою спеціальних інструментів.	
Системне тестування	Перевірка працездатності ізольованої частини функціоналу системи (без інтеграції).	
Інтеграційно-функціональне тестування	Перевірка працездатності наскрізних бізнес-операцій між різними системами для нового та для раніше розробленого функціоналу.	Інженер тестування
Навантажувальне тестування	Перевірка працездатності системи при одночасній роботі більшої кількості користувачів. Виконується тільки для високонавантажених систем та бізнес-операцій	
Динамічне тестування застосунку на безпеку	Перевірка системи на вразливості (підвищення привілеїв, виконання довільного коду, sql injection,...) та «закладки»	



Циклічний автоматизований процес на середовищах організації

- Безшовний наскрізний процес виробництва ПЗ
- Максимальна автоматизація на кожному етапі процесу
- Автоматизована збірка та розгортання дистрибутива
- Автоматичне тестування, автоматичний аналіз якості коду, перевірки коду та застосунків на ІБ
- Автоматизоване управління інфраструктурою, інфраструктура як код, динамічне виділення інфраструктури з хмари





Основні практики DevOps



Безперервна інтеграція (CI, Continuous Integration) – це практика розробки програмного забезпечення, яка полягає у виконанні частих автоматизованих збірок для якнайшвидшого виявлення помилок. Це дозволяє бути впевненим у тому, що код перебуває у робочому стані. Перехід до безперервної інтеграції дозволяє знизити трудомісткість інтеграції змін коду з існуючою кодовою базою і зробити її більш передбачуваною за рахунок найбільш раннього виявлення та усунення помилок та протиріч, зробити процес розробки прозорим для всіх учасників команди.

Безперервна доставка (CD_L, Continuous delivery) – практика розробки програмного забезпечення, що гарантує те, що програмне забезпечення завжди готове до розгортання промислової експлуатації. Практика включає автоматизацію розгортання на тестові середовища, автоматизацію тестування успішності розгортання на рівні адміністраторів (технічні та функціональні тести розгортання), а також інтеграцію з процесами автоматизованого функціонального, навантажувального тестування та динамічного тестування на безпеку.



Безперервне розгортання (CD_P, Continuous deployment) – практика розробки програмного забезпечення, спрямована на повну автоматизацію доставки від середовища розробки у промислове середовище. Є подальшим розвитком процесу безперервної доставки. В умовах організації додатково зачіпає процес реєстрації у сховищі дистрибутивів організації та автоматизацію приймальних випробувань. Характеризується тісною взаємодією розробки та експлуатації.



Проблеми, що вирішуються



Існуючі проблеми виробництва

- 1 Через відсутність загальних стандартів ведення розробки (патерни програмування, нотації тощо) та централізованого зберігання коду трапляються конфлікти версій, пишеться неоптимальний код, знижується ефективність розробки.
- 2 Розробка не виконує інтеграційні тести, тому помилки інтеграції виявляються пізніх етапах -до ПВ.
- 3 Ручне керування середовищами та розгортанням призводить до тривалого очікування стендів командами (до кількох місяців) та збільшує кількість помилок при налаштуванні стендів.
- 4 Великий обсяг ручного тестування, відсутність модульних та швидких "smoke" тестів збільшує час тестування або скорочує обсяг тестового покриття.
- 5 Перевірка якості та безпеки коду при складанні не є обов'язковою. Тому в результаті можемо отримати небезпечний (з уразливістю) та неоптимальний код (збільшення навантаження на супровід).

Вирішення за допомогою DevOps

 VC	Код зберігається централізовано, будь-яка зміна зберігається з новою версією. Розробка ведеться згідно зі стандартами, адаптованими до команди.
 AT	При кожній збірці виконуються швидкі smoke тести, що перевіряють інтеграцію "на заглушках" та емуляторах, помилки виявляються відразу при складанні.
 AD	Виділення середовища відбувається за хвилини з хмари організації. Розгортання та конфігурування автоматизовано.
 IaaS	
 DI	
 AT	Весь код покритий модульними тестами, тестування є максимально автоматизованим.
 ACR	Код і готовий застосунок перевіряються на вразливості та на якість - як в Apple і Google всі події централізовано журналюються.
 PrCR	
 IS	
 JaaS	



На ринку присутні десятки інструментів DevOps

PERIODIC TABLE OF DEVOPS TOOLS																		2 Fm Aws Amazon Web Services																													
1 Fm Gh Github																																															
3 Os Gt Git		4 Pd Dm DBmaestro																5 En Ch Chef		6 En Pu Puppet		7 Os An Ansible		8 En Sl Salt		9 Os Dk Docker		10 Pd Az Azure																			
11 Fm Bb Bitbucket		12 Os Lb Liquibase																13 Os Ot Otto		14 En Bl BladeLogic		15 Os Va Vagrant		16 Fr Tf Terraform		17 Os Rk rkt		18 En Gc Google Cloud Platform																			
19 Os Gl GitLab		20 En Rg Redgate		21 Os Mv Maven		22 Os Gr Gradle		23 Os At ANT		24 Os Fn FitNesse		25 Fr Se Selenium		26 Os Ga Gatling		27 Fr Dh Docker Hub		28 Os Jn Jenkins		29 Pd Ba Bamboo		30 Os Tr Travis CI		31 Pd Gd Deployment Manager		32 Os Sf SmartFrog		33 Os Cn Consul		34 Os Bc Bcf92		35 Os Mo Mesos		36 En Rs Rackspace													
37 Os Sv Subversion		38 En Dt Datical		39 Os Gt Grunt		40 Os Gp Gulp		41 Os Br Broccoli		42 Fr Cu Cucumber		43 Os Cj Cucumber.js		44 Fr Qu Qunit		45 Os Npm npm		46 Fm Cs Codeship		47 Pd Vs Visual Studio		48 Fm Cr CircleCI		49 Fr Cp Capistrano		50 Fr Ju JuJu		51 Os Rd Rundeck		52 Os Cf CFEngine		53 Fr Ds Swarm		54 Os Op OpenStack													
55 Os Hg Mercurial		56 En Dp Delphix		57 Fr Sb sbt		58 Os Mk Make		59 Os Ck CMake		60 Fr Ju JUnit		61 Fr Jm JMeter		62 Fr Tn TestNG		63 Os Ay Artifactory		64 Fm Tc TeamCity		65 Fm Sh Shippable		66 Os Cc CruiseControl		67 En Ry RapidDeploy		68 Fm Cy CodeDeploy		69 En Oc Octopus Deploy		70 En No CA Nolio		71 Os Kb Kubermetes		72 Fm Hr Heroku													
73 En Cw ISPW		74 En Id Idera		75 Os Msb MSBuild		76 Os Rk Rake		77 Fr Pk Packer		78 Os Mc Mocha		79 En Xltv XL TestView		80 Os Jm Jasmine		81 Os Nx Nexus		82 Os Co Continuum		83 Fm Ca Continua CI		84 Pd So Solano CI		85 En Xld XL Deploy		86 En EB ElectricBox		87 Fm Dp Deploybot		88 En Ud UrbanCode Deploy		89 Os Nm Nomad		90 En Os OpenShift													
																		91 En Xlr XL Release		92 En Ur UrbanCode Release		93 En Bm BMC Release Process		94 En Hp HP Codar		95 En Au Automic		96 En Pl Plutora Release		97 En Sr Serena Release		98 Pd Tfs Team Foundation		99 Fm Tr Trello		100 Pd Jr Jira		101 Fm Rf HipChat		102 Fm Sl Slack		103 Fm Fd Flowdock		104 Pd Pv Pivotal Tracker		105 En Sn ServiceNow	
																		106 Os Ki Kibana		107 Fm Nr New Relic		108 Os Ni Nagios		109 Os Zb Zabbix		110 En Dd Datadog		111 Os El Elasticsearch		112 Os Ss StackState		113 En Sp Splunk		114 Fm Le Logentries		115 Fm Sl Sumo Logic		116 Os Ls Logstash		117 Os Gr Graylog		118 Os Sn Snort		119 Os Tr Tripwire		120 En Ff Fortify	

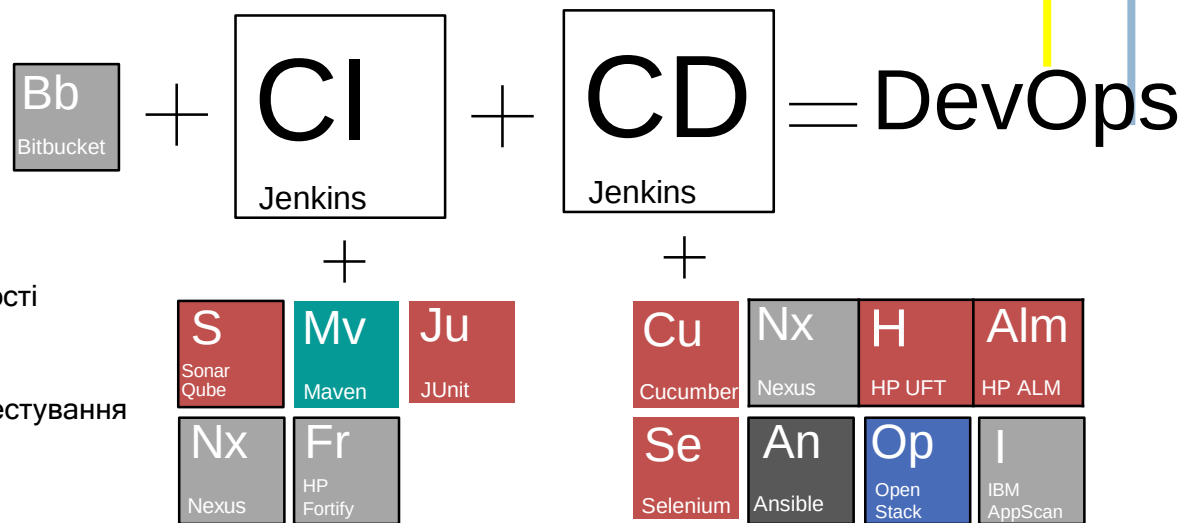
Джерело: <https://xebialabs.com/periodic-table-of-devops-tools/>



Інструменти: затверджені та ті, що обираються командами

Затверджені інструменти:

- **Atlassian Bitbucket** – система версійного зберігання коду
- **Maven** – інструмент авто-збірки
- **JUnit** – бібліотека для модульного тестування Java застосунків
- **Jenkins** – інструмент управління процесом збірки, розгортання та налаштування
- **Sonar, HP Fortify, IBM AppScan** – перевірка коду на якість та вразливості
- **HP ALM** – інструмент управління процесом тестування
- **HP UFT, Selenium, Cucumber, HP PC** – автотести, навантажувальне тестування
- **Ansible** – управління конфігураціями та налаштуванням середовищ
- **Nexus** – управління зберіганням дистрибутивів
- **OpenStack** – віртуалізація інфраструктури, приватна хмара



Інструменти, що обираються командами:

□ - Цільовий інструмент, затверджений/ рекомендований до використання

Однак є інструменти, тісно пов'язані з певною технологією. Такі інструменти не затверджуються на архітектурній раді, їх вибір здійснюється робочою групою:

- Командою розробки
- ЦЕ процесів виробництва (ЦЕ DevOps)
- Командою сервіса CI або CD
- Прикладними та системними адміністраторами



Чому нам важливо мати стандарти у виборі інструментів DevOps?

- Централізований супровід інструментів дозволяє командам зосередитись на розробці
- Надання централізованого сервісу 24x7
- Об'єктивний вибір інструментів з урахуванням критеріїв вартості (в т.ч. вартість експлуатації), ефективності та надійності та відповідність інструментів принципам прозорості
- Наявність безлічі інструментів*, що виконують однакову функцію, створює «вузьке місце» на стороні Ops, що суперечить принципам DevOps.
- До централізованих інструментів додається централізований моніторинг, який дозволяє простежити весь шлях релізу від розробки до ПРОДУ.



DevOps Toolchain

Безперервна інтеграція

1. Кодування



Розробник програмує код і пише модульні тести до цього коду

Інструменти: Bitbucket JUnit



2. Управління збіркою



Після збереження коду в **Bitbucket** запускається workflow процес збірки в **Jenkins**. **Jenkins** запускає модульні тести, статичний аналіз якості коду та ІБ. **Ansible** здійснює автоматизоване розгортання дистрибутива на середовище розробки.

Інструменти: Nexus Bitbucket
 Jenkins ANSIBLE Інструменти збірки FORTIFY sonarqube

Безперервна доставка

3. Підготовка середовища



Після складання дистрибутива запускається workflow процесу розгортання в **Jenkins**, відбувається (опціонально) виділення тестових середовищ, готуються тестові набори даних, здійснюється автоматичне конфігурування середовищ через **Ansible**.

Інструменти: Jenkins ANSIBLE Nexus

4. Управління розгортанням



Ansible розподілює дистрибутив на тестові середовища, виконуються технічні та функціональні тести розгортання

Інструменти: Jenkins ANSIBLE Nexus

5. Управління тестуванням



Jenkins викликає **HP ALM** і запускається набір тестів як автоматизованих (API, GUI, інтеграційних, безпеки тощо), так і «ручних»

Інструменти: hp ALM Unified Functional Testing (UFT) AppScan IBM Security

Моніторинг та зворотній зв'язок



Механізм оповіщень та моніторингу гарантує швидкий цикл зворотного зв'язку. Доступний для всіх етапів

Інструменти: elastic QlikView

Безперервне розгортання

6. ПВ



У разі успішного проходження випробувань приймається рішення (людина) про розгортання дистрибутива на ПРОД середовище

7. Розгортання на ПРОД



Відбувається автоматизоване розгортання дистрибутива на середовище ПРОД. У разі помилок - виконується автоматичний відкат (Rollback)

Інструменти: Jenkins ANSIBLE Nexus



Нова роль – DevOps інженер

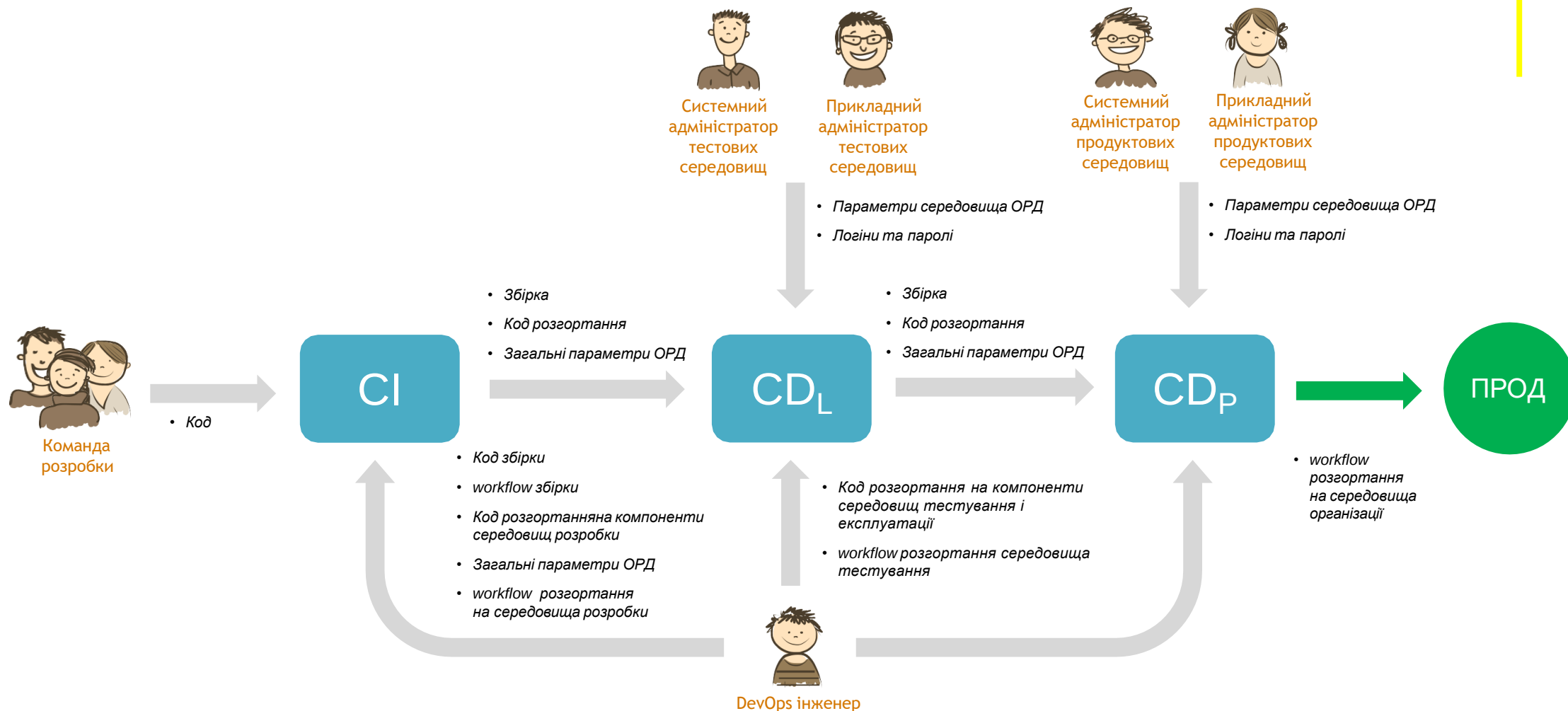


DevOps інженер – є експертом з практик DevOps, володіє інструментами CI і CD, вибудовує Pipeline збірки та розгортання.

- Налаштовує процес CI на централізованому сервісі, інтегрує його з GitFlow та кодом розгортання на середовищах розробки.
- Розробляє процес постачання ПЗ на цільових інструментах CD для всіх середовищ (розробки, тестування та експлуатації).
- Забезпечує однаковість процесів доставки та вибудовує їх у єдиний безшовний процес доставки.
- Керує та забезпечує процес поширення змін тестових середовищ для середовищ розробки та тестування (за межами обов'язків середовища організації).
- Надає супровід прикладним та системним адміністраторам середовищ організації
- Має інструменти централізованого сервісу CI і CD, а також командні інструменти, в т.ч. з юніт-тестування та збірк.
- Має знання технічної та інтеграційної архітектури, механізмів переносів налаштувань між середовищами.
- Має базові навички адміністрування прикладного і системного ПЗ.



Робота DevOps Toolchain



Донецький національний технічний університет



Тема 2. Основи застосування системи контролю версій Git. Лекція 2. Системи керування версіями

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

27

- ☐ Мета лекції
- ☐ Основна термінологія
- ☐ Системи керування версіями
- ☐ Вступ до Git



Мета лекції

28

- ☐ Розглянути види систем керування версіями
- ☐ Ознайомитися з системою керування версіями Git



Система контролю версій

29

- ❑ Система контролю версій (СКВ) — це система, яка реєструє зміни в одному або кількох файлах для того, щоб надалі була можливість повернутися до певних старих версій цих файлів.



Які варіанти?

30

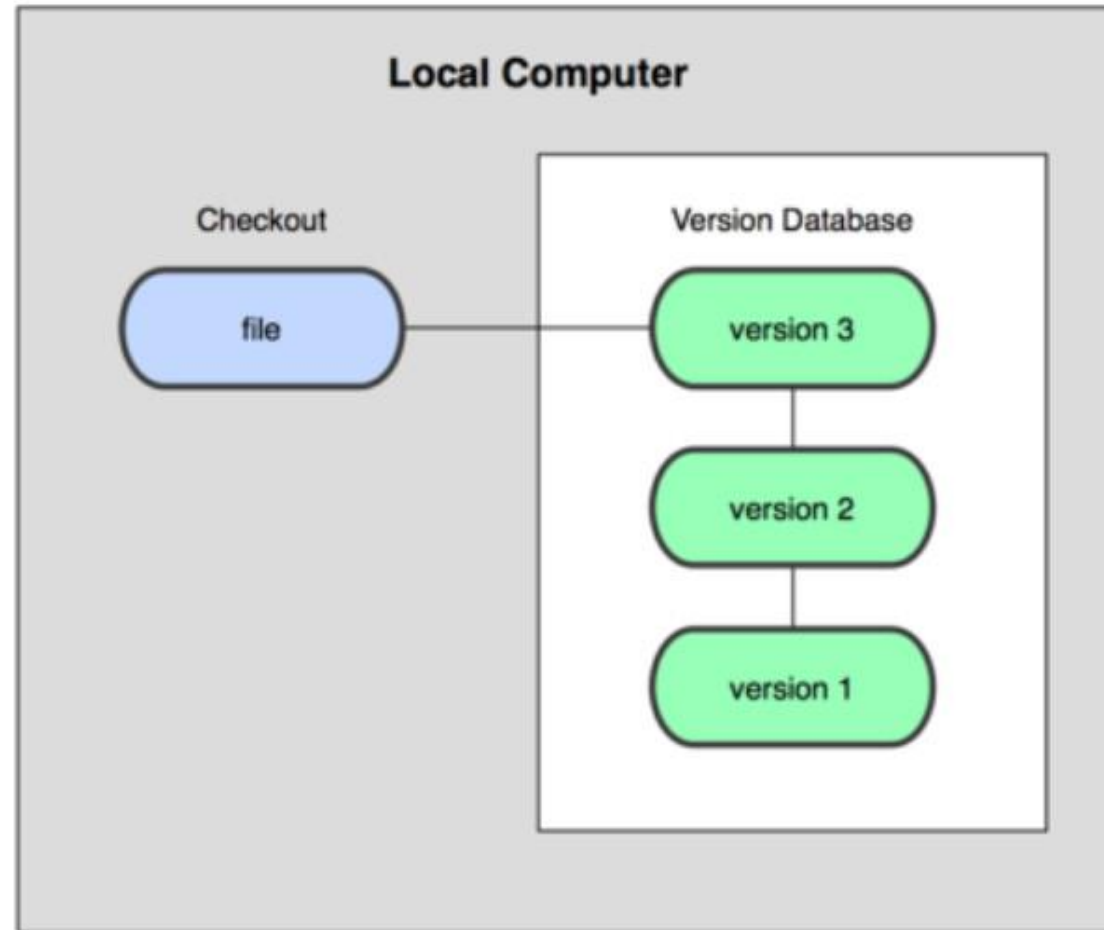
```
site.php
site.php_01.01.2000
site.php_02.01.2000
site.php_03.01.2000
site.php_04.01.2000
site.php_05.01.2000
site.php_06.01.2000
site.php_07.01.2000
```

```
▼ 01.01.2000/
  site.php
▼ 02.01.2000/
  site.php
▼ 03.01.2000/
  site.php
▶ 04.01.2000/
▶ 05.01.2000/
▶ 06.01.2000/
▶ 07.01.2000/
```

```
▼ 01.01.2000/
  site.php_11:00:00
  site.php_15:00:00
▼ 02.01.2000/
  site.php_01:00:00
  site.php_03:00:00
  site.php_17:00:00
▶ 03.01.2000/
▶ 04.01.2000/
▶ 05.01.2000/
▶ 06.01.2000/
▶ 07.01.2000/
```



Локальні системи контролю версій





RCS (Revision Control System)

32

□ Приклад сесії

- ci file.txt
- co file.txt
- ci file.txt



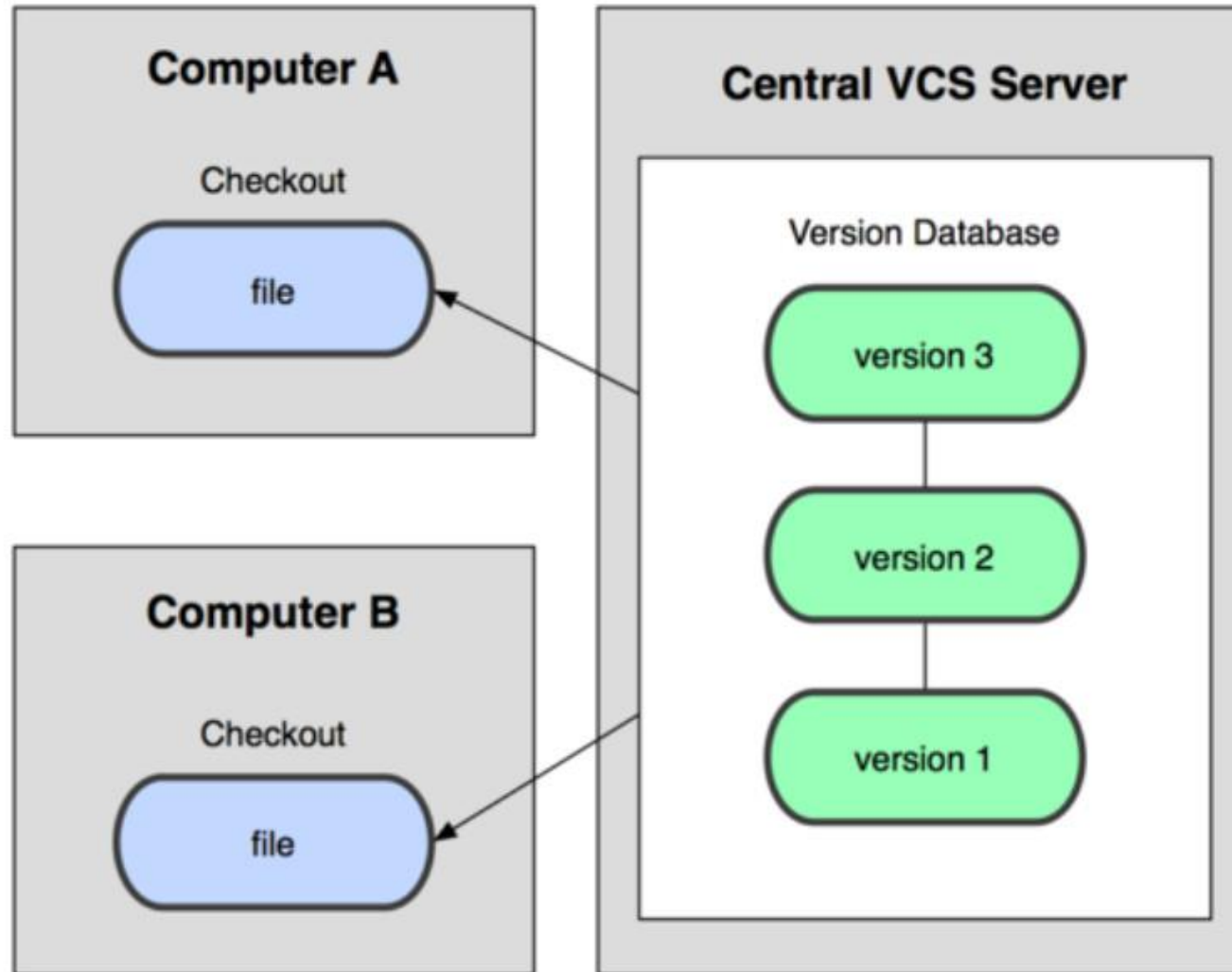
Недоліки RCS

- ☐ Робота лише з одним файлом, кожен файл має контролюватись окремо
- ☐ Незручний механізм одночасної роботи кількох користувачів із системою, сховище просто блокується, поки користувач, що його заблокував, не розблокує його
- ☐ Від бекапів вас ніхто не звільняє, ви ризикуєте втратити все



Централізовані системи контролю версій

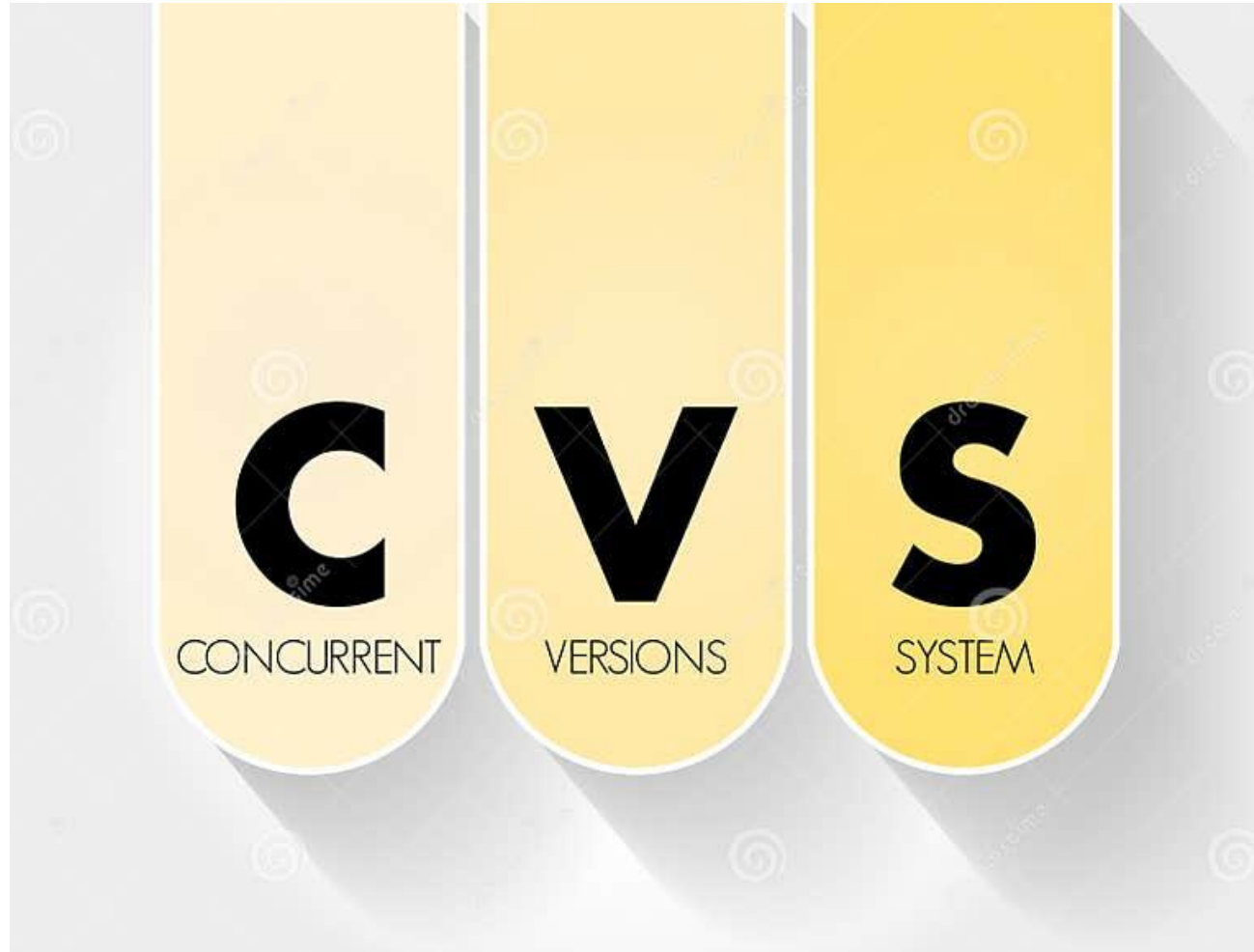
34





CVS (Concurrent Versions System)

35





Приклад сесії CVS

1. `cd some-project`
2. `cvs import -m "New project" path-in-repository none start`
3. `cd some-working-dir`
4. `cvs checkout path-in-repository`
5. `cvs commit -m "Some changes"`
6. `cvs update`



Недоліки CVS

- ☐ Немає можливості зберігати версії директорій.
- ☐ Стандартний спосіб обійти цю перешкоду – зберегти будь-який файл (наприклад, README.txt) в директорії;
- ☐ Переміщення або перейменування файлів не піддається контролю версій.
- ☐ Стандартний спосіб зробити це: спочатку скопіювати файл, видалити старий за допомогою команди `cv remove`, а потім додати його з новим ім'ям за допомогою команди `cv add`;



Subversion (SVN)

❑ Переваги у порівнянні з CVS:

- Атомарне внесення змін (commit). Якщо обробка коміту була перервана, не буде внесено жодних змін.
- Перейменування, копіювання та переміщення файлів зберігає всю історію змін.
- Директорії, символічні посилання та мета-дані схильні до контролю версій.
- Ефективне збереження змін для бінарних файлів.

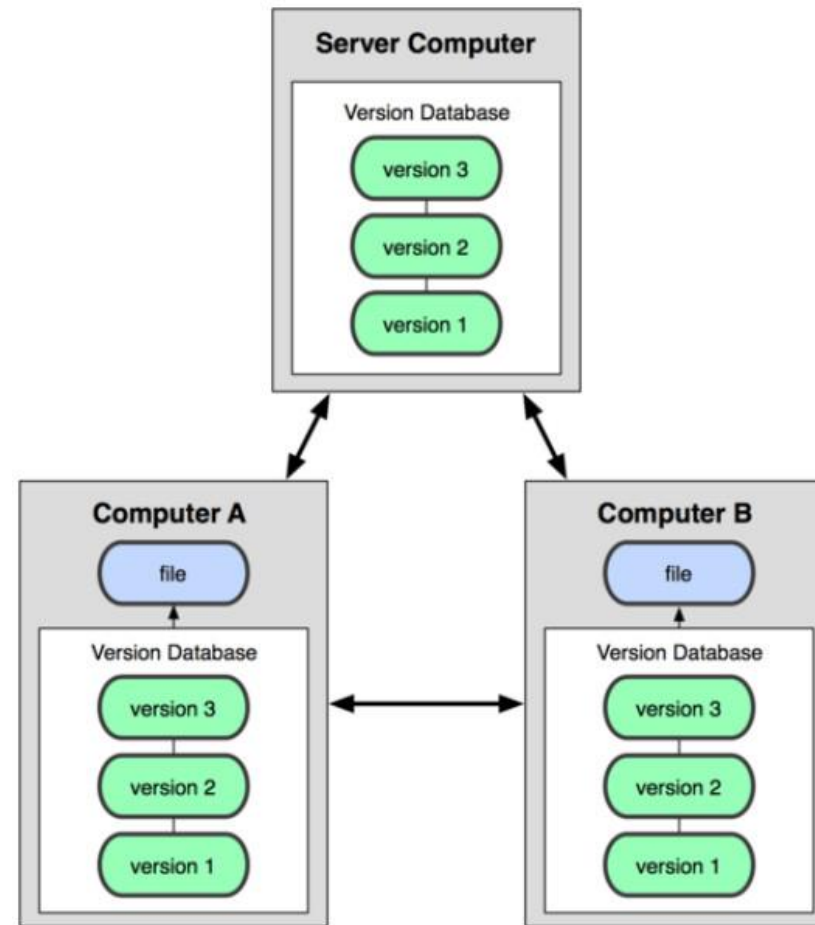


Приклад сесії SVN

- ☐ `cd some-project`
- ☐ `svn import -m "New project" path-in-repository`
- ☐ `cd some-working-dir`
- ☐ `svn checkout path-in-repository`
- ☐ `svn commit -m "Some changes"`
- ☐ `svn update (up)`



Розподілені системи контролю версій





Розподілені СКВ

- ☐ Git (<http://git-scm.com/>)
- ☐ Mercurial (<http://www.selenic.com/mercurial/>)
- ☐ Bazaar (<http://bazaar-vcs.org/>)
- ☐ Codeville (<http://codeville.org/>)
- ☐ Darcs (<http://darcs.net/>)
- ☐ Monotone (<http://monotone.ca/>)



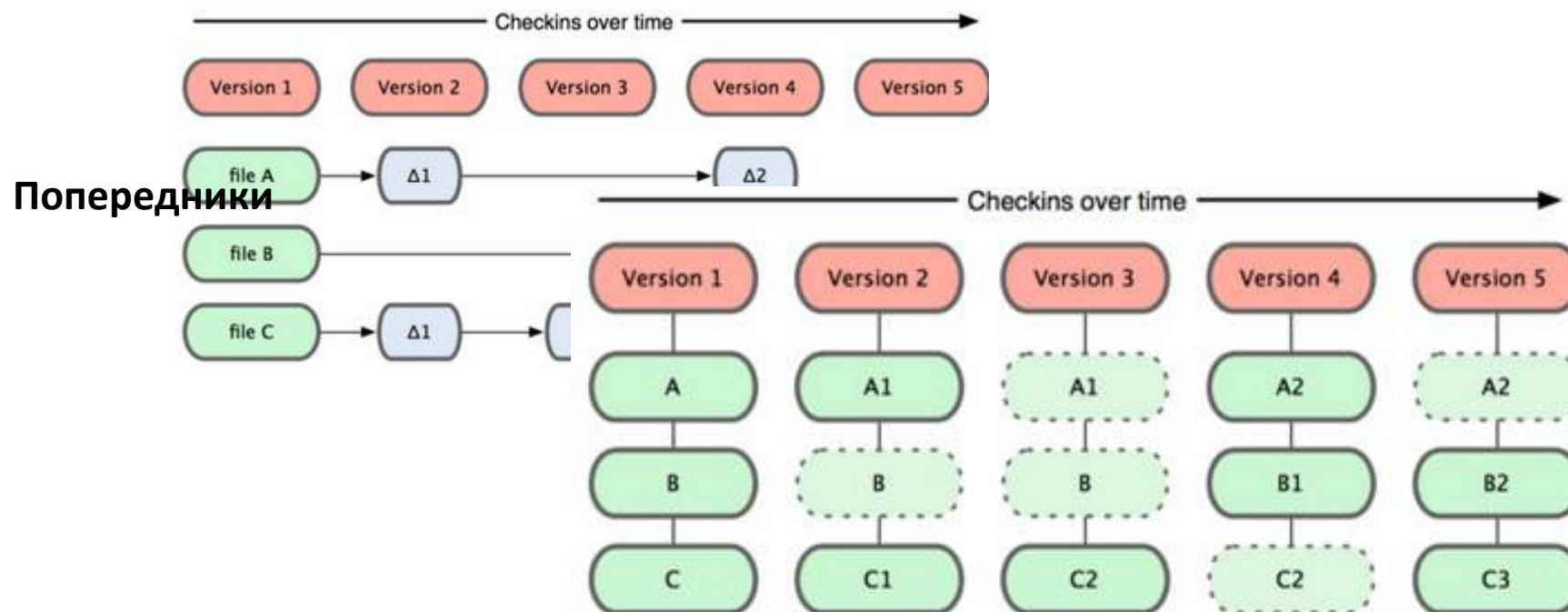
Коротка історія Git

❑ Основні вимоги до нової системи:

- швидкість
- простота дизайну
- підтримка нелінійної розробки (тисячі паралельних гілок)
- повна розподіленість
- можливість ефективної роботи з такими великими проектами, як ядро Linux (як за швидкістю, так і за розміром даних)



Зліпки замість патчів



Git



Відмінності/вимоги Git1

☐ Майже всі операції локальні



Відмінності/вимоги Git₂

- ☐ Майже всі операції локальні
- ☐ Git стежить за цілісністю даних



Відмінності/вимоги Git₃

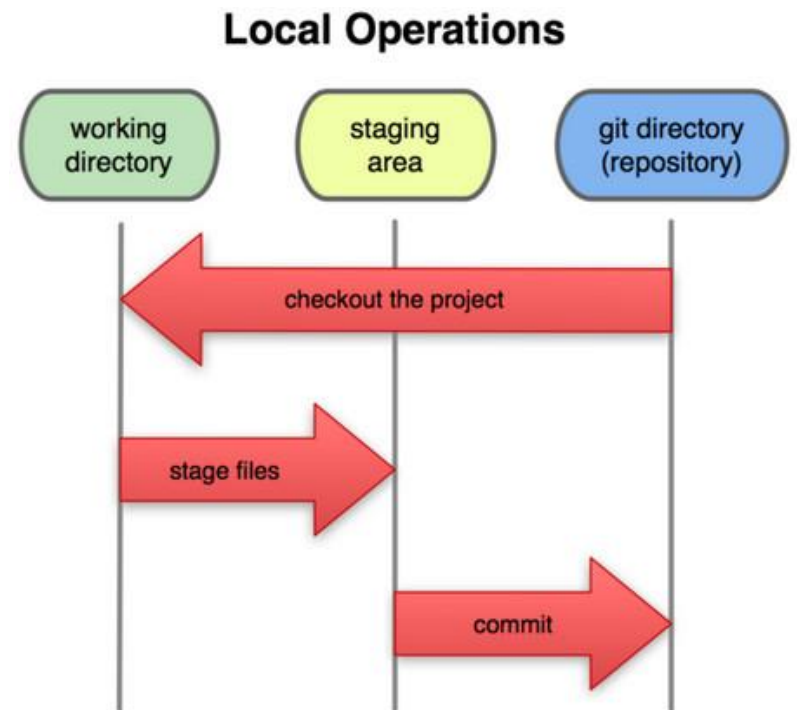
46

- ☐ Майже всі операції локальні
- ☐ Git стежить за цілісністю даних
- ☐ Найчастіше дані в Git тільки додаються



Відмінності/вимоги Git₄

- ❑ Майже всі операції локальні
- ❑ Git стежить за цілісністю даних
- ❑ Найчастіше дані в Git тільки додаються
- ❑ Три стани:
 - каталог Git'a (Git directory),
 - робочий каталог (working directory),
 - область підготовлених файлів (staging area).





Стандартний процес Git

- ☐ Внесення змін до файлів у своєму робочому каталозі
- ☐ Підготовка файлів, додаючи їх зліпки до області підготовлених файлів
- ☐ Виконання коміту, який бере підготовлені файли з індексу та поміщає їх у каталог Git'a на постійне зберігання



Інсталяція Git₁. Вихідні коди

- ☐ Встановлення зібраного пакету для вашої платформи
- ☐ Інсталяція з вихідних джерел
 - curl
 - zlib
 - openssl
 - expat
 - libiconv



Інсталяція Git₂. Linux

- ☐ yum install git-core
- ☐ apt-get install git



Інсталяція Git₃. MacOS

- ❑ <http://code.google.com/p/git-osx-installer>
- ❑ `sudo port install git-core +svn +doc +bash_completion +gitweb`



Інсталяція Git₄. Windows

□ <https://gitforwindows.org/>



Початкове налаштування Git₁

□ git config

□ Види конфігурації

- загальні для всіх користувачів системи (git config --system)
- налаштування для конкретного користувача (git config --global)
- конфігураційний файл у каталозі Git'a (git config)



Початкове налаштування Git₂

54

□ Дані користувача

- `git config --global user.name "John Doe"`
- `git config --global user.email johndoe@example.com`



Початкове налаштування Git₃

☐ Вибір редактора

- `git config --global core.editor emacs`



Початкове налаштування Git₄

□ Утиліта порівняння

- `git config --global merge.tool vimdiff`
- `git config --global merge.tool kdiff3`
- `git config --global merge.tool tkdiff`
- `git config --global merge.tool meld`
- `git config --global merge.tool xxdiff`
- `git config --global merge.tool emerge`
- `git config --global merge.tool gvimdiff`
- `git config --global merge.tool ecmerge`
- `git config --global merge.tool opendiff`



Перевірка налаштувань Git

❑ git config --list

- user.name=Iaroslav Dorohyi
- user.email=cisco.rna@gmail.com
- ...
- color.status=auto
- color.diff=auto

❑ git config user.name



Як отримати допомогу в Git?

58

- ☐ `git help <команда>`
- ☐ `git <команда> --help`
- ☐ `man git-<команда>`



Створення нового репозиторію

- ☐ \$ cd some-project-path
- ☐ \$ git init
- ☐ \$ git add *.c
- ☐ \$ git add README
- ☐ \$ git add .
- ☐ \$ git commit

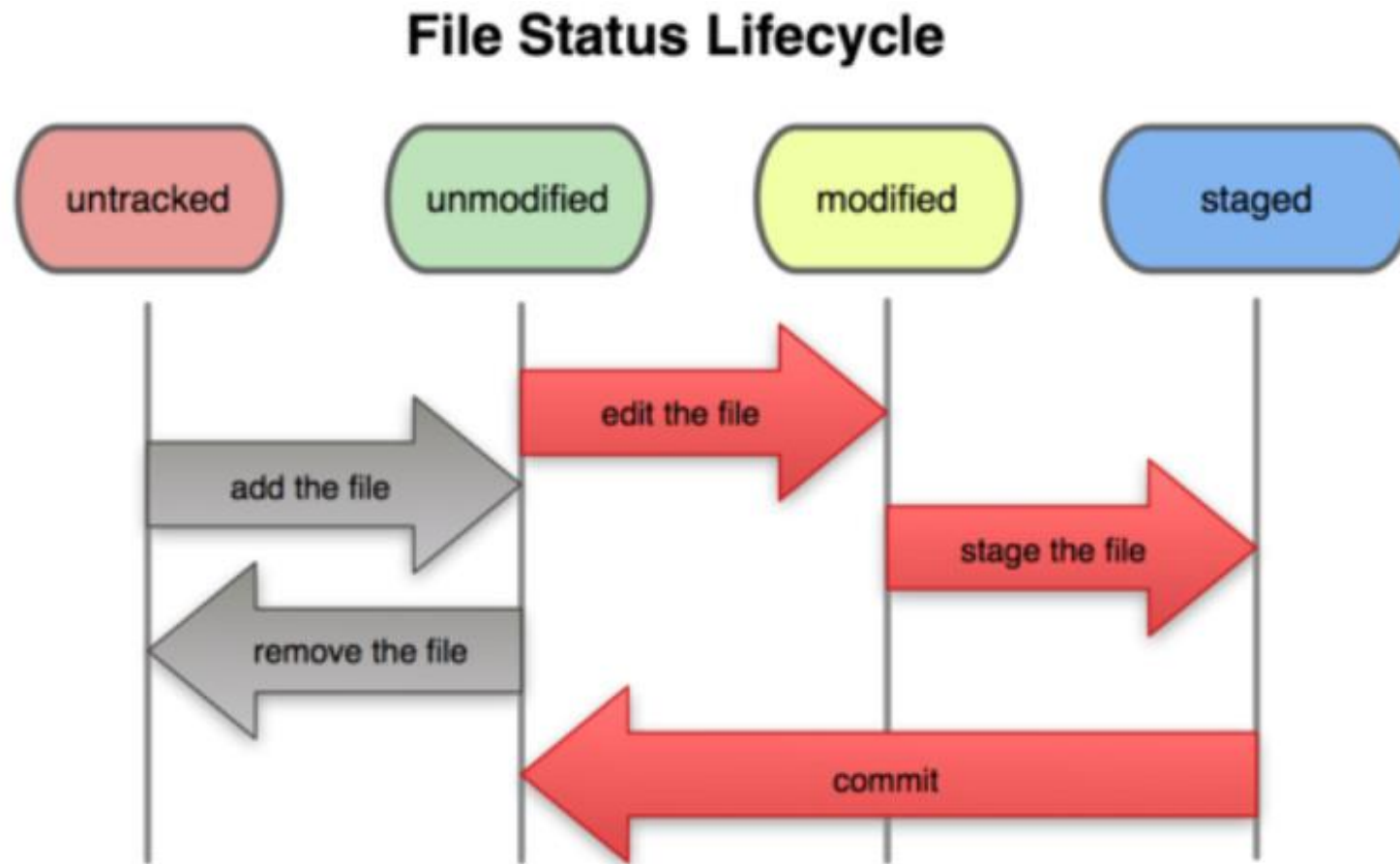


Клонування існуючого репозиторію

- ❑ `$ git clone [options] [--] repo [dir]`
- ❑ `$ git clone git://github.com/libgit2/rugged.git`
- ❑ `$ git clone git://github.com/libgit2/rugged.git myrugged`



Запис змін до репозиторію





Визначення стану файлів

❑ git status

- untracked
- modified
- added to commit

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

```
$ touch README
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       README
nothing added to commit but untracked files present (use "git add" to track)
```



Відстеження нових файлів

63

- ☐ git add README
- ☐ git status

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
```



Індексація змінених файлів₁

❑ Перевіряємо статус

- `git status`

❑ Додаємо файл і знову перевіряємо статус

- `git add benchmarks.rb`
- `git status`

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   benchmarks.rb
#
```

```
$ git add benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
#
```



Індексція змінених файлів₂

❑ Редагуємо файл і перевіряємо статус

- vim benchmarks.rb
- git status

❑ Додаємо файл і перевіряємо статус

- git add benchmarks.rb
- git status

```
$ vim benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   benchmarks.rb
#
```

```
$ git add benchmarks.rb
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
#
```



Ігнорування файлів

❑ .gitignore

- порожні рядки і рядки, що починаються з #, ігноруються
- можна використовувати стандартні glob шаблони
- шаблон можна закінчити символом слеша (/) для вказівки каталогу
- можна інвертувати шаблон, використовуючи знак оклику (!) як перший символ

```
$ cat .gitignore
*.[oa]
*~
```



Перегляд індексованих та неіндексованих змін₁

67

❑ git status

❑ git diff

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#
#       modified:   benchmarks.rb
#
```

```
$ git diff
diff --git a/benchmarks.rb b/benchmarks.rb
index 3cb747f..da65585 100644
--- a/benchmarks.rb
+++ b/benchmarks.rb
@@ -36,6 +36,10 @@ def main
     @commit.parents[0].parents[0].parents[0]
   end

+  run_code(x, 'commits 1') do
+    git.commits.size
+  end
+
   run_code(x, 'commits 2') do
     log = git.commits('master', 15)
     log.size
```



Перегляд індексованих та неіндексованих змін₂

- ☐ git diff - cached
- ☐ git add benchmarks.rb
- ☐ echo '# test line' >> benchmarks.rb
- ☐ git status

```
$ git diff --cached
diff --git a/README b/README
new file mode 100644
index 00000000..03902a1
--- /dev/null
+++ b/README2
@@ -0,0 +1,5 @@
+grit
+ by Tom Preston-Werner, Chris Wanstrath
+ http://github.com/mojombo/grit
+
+Grit is a Ruby library for extracting information from a Git repository
```

```
$ git add benchmarks.rb
$ echo '# test line' >> benchmarks.rb
$ git status
# On branch master
#
# Changes to be committed:
#
#       modified:   benchmarks.rb
#
# Changes not staged for commit:
#
#       modified:   benchmarks.rb
#
```




Перегляд індексованих та неіндексованих змін₃

❑ git diff

❑ git diff - cached

```
$ git diff
diff --git a/benchmarks.rb b/benchmarks.rb
index e445e28..86b2f7c 100644
--- a/benchmarks.rb
+++ b/benchmarks.rb
@@ -127,3 +127,4 @@ end
  main()

  ##pp Grit::GitRuby.cache_client.stats
  +# test line
```

```
$ git diff --cached
diff --git a/benchmarks.rb b/benchmarks.rb
index 3cb747f..e445e28 100644
--- a/benchmarks.rb
+++ b/benchmarks.rb
@@ -36,6 +36,10 @@ def main
    @commit.parents[0].parents[0].parents[0]
  end

+  run_code(x, 'commits 1') do
+    git.commits.size
+  end
+
  run_code(x, 'commits 2') do
    log = git.commits('master', 15)
    log.size
```



Фіксація змін

70

❑ git commit

❑ git commit -m "Story 182: Fix benchmarks for speed"

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   README
#       modified:   benchmarks.rb
#
~
~
~
".git/COMMIT_EDITMSG" 10L, 283C
```

```
$ git commit -m "Story 182: Fix benchmarks for speed"
[master]: created 463dc4f: "Fix benchmarks for speed"
2 files changed, 3 insertions(+), 0 deletions(-)
create mode 100644 README
```



Ігнорування індексації

- ☐ git status
- ☐ git commit -a -m 'added new benchmarks'

```
$ git status
# On branch master
#
# Changes not staged for commit:
#
#       modified:   benchmarks.rb
#
$ git commit -a -m 'added new benchmarks'
[master 83e38c7] added new benchmarks
1 files changed, 5 insertions(+), 0 deletions(-)
```



Видалення файлів₁

❑ Просте видалення

- `rm grit.gemspec`
- `git status`

❑ Видалення засобами Git

- `git rm grit.gemspec`
- `git status`

```
$ rm grit.gemspec
$ git status
# On branch master
#
# Changes not staged for commit:
#   (use "git add/rm <file>..." to update what will be committed)
#
#       deleted:    grit.gemspec
#
```

```
$ git rm grit.gemspec
rm 'grit.gemspec'
$ git status
# On branch master
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       deleted:    grit.gemspec
#
```



Видалення файлів₂

- ☐ `git rm --cached readme.txt`
- ☐ `git rm log/\*.log`
- ☐ `git rm \*~`



Переміщення файлів

- ❑ git mv file_from file_to
 - git mv README.txt README
 - \$ git status
- ❑ Еквівалент
 - mv README.txt README
 - \$ git rm README.txt
 - \$ git add README

```
$ git mv README.txt README
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    README.txt -> README
#
```



Перегляд історії комітів₁

- ☐ git clone
git://github.com/schacon/simplegit-progit.git
- ☐ git log
- ☐ git log -p -2

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700

    changed the version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 10:31:28 2008 -0700

    first commit
```

```
$ git log -p -2
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700

    changed the version number

diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,5 @@ require 'rake/gempackagetask'
 spec = Gem::Specification.new do |s|
   s.name      = "simplegit"
-  s.version   = "0.1.0"
+  s.version   = "0.1.1"
   s.author    = "Scott Chacon"
   s.email     = "schacon@gee-mail.com"

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

diff --git a/lib/simplegit.rb b/lib/simplegit.rb
index a0a60ae..47c6340 100644
--- a/lib/simplegit.rb
+++ b/lib/simplegit.rb
@@ -18,8 +18,3 @@ class SimpleGit
   end

end

- if $0 == __FILE__
-   git = SimpleGit.new
-   puts git.show
- end
\ No newline at end of file
```




Перегляд історії комітів₂

- ❑ git log -U1 --word-diff
- ❑ git log --stat

```
$ git log -U1 --word-diff
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700

    changed the version number

diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -7,3 +7,3 @@ spec = Gem::Specification.new do |s|
   s.name       = "simplegit"
   s.version    = ["0.1.0"]{+"0.1.1"+}
   s.author     = "Scott Chacon"
```

```
$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700

    changed the version number

Rakefile | 2 +-
1 files changed, 1 insertions(+), 1 deletions(-)

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

lib/simplegit.rb | 5 -----
1 files changed, 0 insertions(+), 5 deletions(-)

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 10:31:28 2008 -0700

    first commit

README | 6 +++++
Rakefile | 23 +++++
lib/simplegit.rb | 25 +++++
3 files changed, 54 insertions(+), 0 deletions(-)
```




Перегляд історії комітів₃

❑ `git log --pretty=oneline`

```
$ git log --pretty=oneline
ca82a6dff817ec66f44342007202690a93763949 changed the version number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 removed unnecessary test code
a11bef06a3f659402fe7563abf99ad00de2209e6 first commit
```

❑ `git log --pretty=format:"%h - %an, %ar : %s"`

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 11 months ago : changed the version number
085bb3b - Scott Chacon, 11 months ago : removed unnecessary test code
a11bef0 - Scott Chacon, 11 months ago : first commit
```

Параметр	Опис даних, що виводяться
%H	Хеш коміта
%h	Скорочений хеш коміта
%T	Хеш дерева
%t	Скорочений хеш дерева
%P	Хеші батьківських комітів
%p	Скорочені хеші батьківських комітів
%an	Ім'я автора
%ae	Електронна пошта автора
%ad	Дата автора (формат відповідає параметру `--date=`)
%ar	Дата автора, відносна (пр. "2 міс. назад")
%cn	Ім'я комітера
%ce	Електронна пошта комітера
%cd	Дата комітера
%cr	Дата комітера, відносна
%s	Коментар

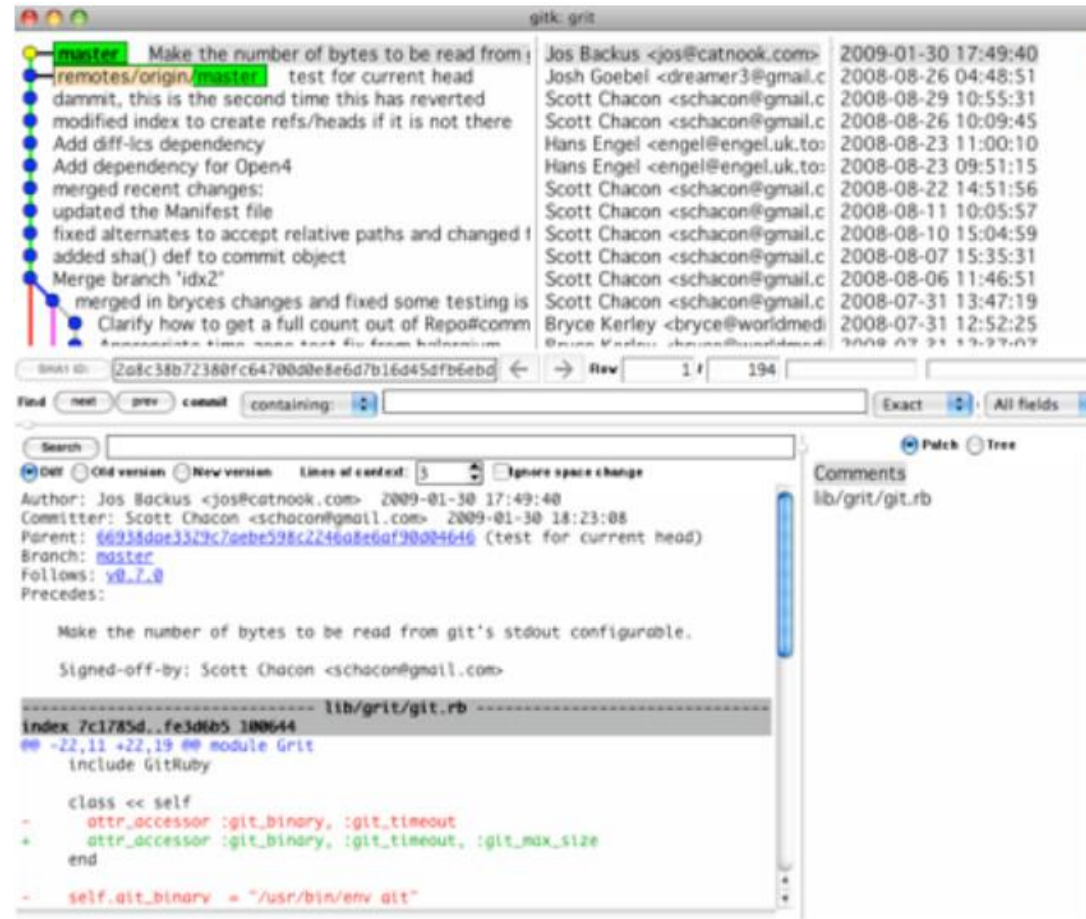


Перегляд історії комітів₄

❑ `git log --since=2.weeks`



Використання графічного інтерфейсу для візуалізації історії





Скасування змін

- ❑ `git commit --amend`
 - `git commit -m 'initial commit'`
 - `git add forgotten_file`
 - `git commit --amend`



Скасування індексації файлу

- ☐ git add .
- ☐ git status

```
$ git add .
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#       modified:   benchmarks.rb
#
```

- ☐ git reset HEAD benchmarks.rb
- ☐ git status

```
$ git reset HEAD benchmarks.rb
benchmarks.rb: locally modified
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   benchmarks.rb
#
```

Донецький національний технічний університет



Тема 2. Основи застосування системи контролю версій Git.

Лекція 3. Git. Віддалені репозиторії. Розгалуження

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

83

- ☐ Мета лекції
- ☐ Робота з віддаленими репозиторіями
- ☐ Розгалуження



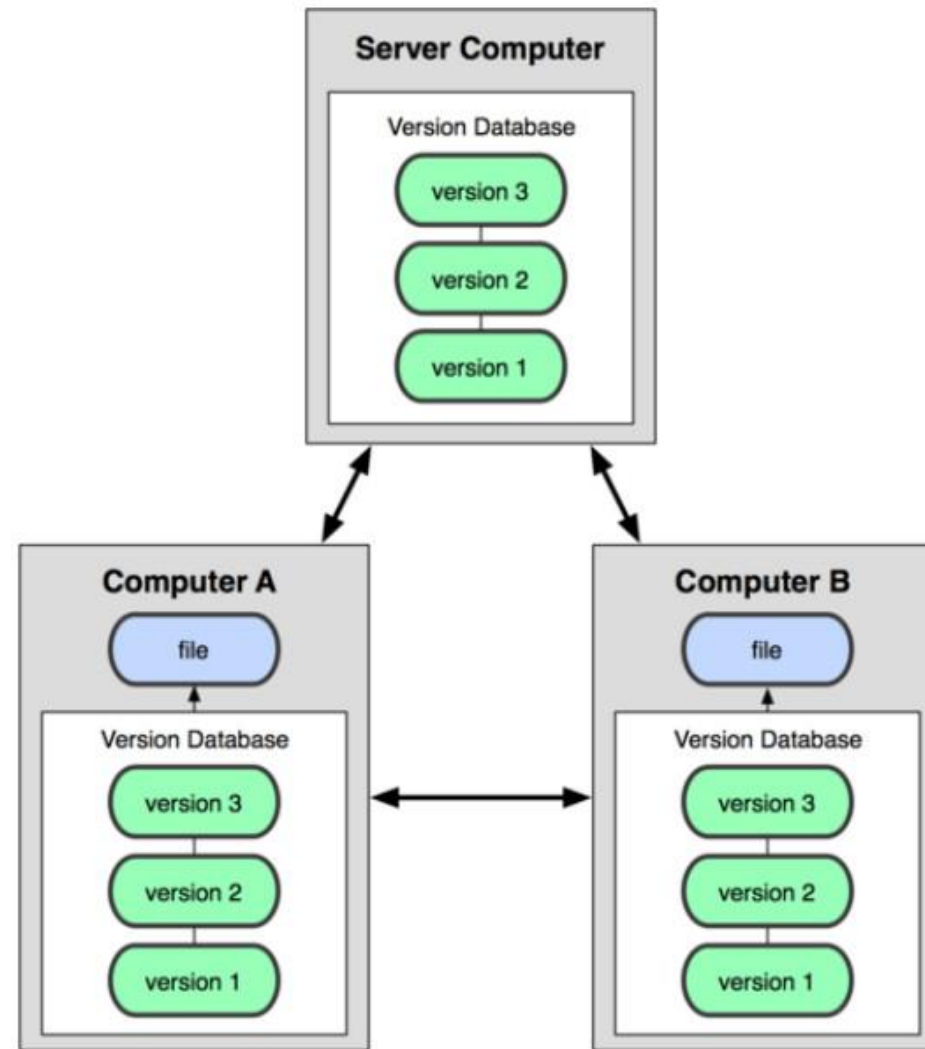
Мета лекції

84

- ☐ Розглянути роботу з віддаленими репозиторіями
- ☐ Розглянути роботу з розгалуженнями в Git



Робота з віддаленими репозиторіями





Відображення віддалених репозиторіїв

- ❑ git clone
git://github.com/schacon/ticgit.git
 - cd ticgit
 - git remote

```
$ git clone git://github.com/schacon/ticgit.git
Initialized empty Git repository in /private/tmp/ticgit/.git/
remote: Counting objects: 595, done.
remote: Compressing objects: 100% (269/269), done.
remote: Total 595 (delta 255), reused 589 (delta 253)
Receiving objects: 100% (595/595), 73.31 KiB | 1 KiB/s, done.
Resolving deltas: 100% (255/255), done.
$ cd ticgit
$ git remote
origin
```

- ❑ git remote -v

```
$ git remote -v
origin git://github.com/schacon/ticgit.git (fetch)
origin git://github.com/schacon/ticgit.git (push)
```

- ❑ cd grit

- ❑ git remote -v

```
$ cd grit
$ git remote -v
bakkdoor git://github.com/bakkdoor/grit.git
cho45 git://github.com/cho45/grit.git
defunkt git://github.com/defunkt/grit.git
koke git://github.com/koke/grit.git
origin git@github.com:mojombo/grit.git
```



Додавання віддалених репозиторіїв

- ☐ git remote
- ☐ git remote add pb git://github.com/paulboone/ticgit.git
- ☐ git remote -v

```
$ git remote
origin
$ git remote add pb git://github.com/paulboone/ticgit.git
$ git remote -v
origin  git://github.com/schacon/ticgit.git
pb      git://github.com/paulboone/ticgit.git
```

- ☐ git fetch pb

```
$ git fetch pb
remote: Counting objects: 58, done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 44 (delta 24), reused 1 (delta 0)
Unpacking objects: 100% (44/44), done.
From git://github.com/paulboone/ticgit
* [new branch]      master    -> pb/master
* [new branch]      ticgit     -> pb/ticgit
```



Fetch

❑ `git fetch [ім'я віддал. сервера]`



Push

☐ git push origin master



Інспекція віддаленого репозиторію

❑ *git remote show origin*

```
$ git remote show origin
* remote origin
  URL: git://github.com/schacon/ticgit.git
  Remote branch merged with 'git pull' while on branch master
    master
  Tracked remote branches
    master
    ticgit
```

```
$ git remote show origin
* remote origin
  URL: git@github.com:defunkt/github.git
  Remote branch merged with 'git pull' while on branch issues
    issues
  Remote branch merged with 'git pull' while on branch master
    master
  New remote branches (next fetch will store in remotes/origin)
    caching
  Stale tracking branches (use 'git remote prune')
    libwalker
    walker2
  Tracked remote branches
    acl
    apiv2
    dashboard2
    issues
    master
    postgres
  Local branch pushed with 'git push'
    master:master
```



Видалення та перейменування віддалених репозиторіїв

☐ ***git remote rename pb paul***

☐ ***git remote***

```
$ git remote rename pb paul  
$ git remote  
origin  
paul
```

☐ ***\$ git remote rm paul***

☐ ***\$ git remote***

```
$ git remote rm paul  
$ git remote  
origin
```



Робота з мітками. Перегляд міток

❑ *git tag*

```
$ git tag  
v0.1  
v1.3
```

❑ *git tag -l 'v1.4.2.*'*

```
$ git tag -l 'v1.4.2.*'  
v1.4.2.1  
v1.4.2.2  
v1.4.2.3  
v1.4.2.4
```




Створення міток

☐ ***git tag -a v1.4 -m 'my version 1.4'***

☐ ***git tag***

```
$ git tag -a v1.4 -m 'my version 1.4'
$ git tag
v0.1
v1.3
v1.4
```

☐ ***git show v1.4***

```
$ git show v1.4
tag v1.4
Tagger: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Feb 9 14:45:11 2009 -0800

my version 1.4
commit 15027957951b64cf874c3557a0f3547bd83b3ff6
Merge: 4a447f7... a6b4c97...
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sun Feb 8 19:02:46 2009 -0800

Merge branch 'experiment'
```



Підписані мітки

❑ ***git tag -s v1.5 -m 'my signed 1.5 tag'***

```
$ git tag -s v1.5 -m 'my signed 1.5 tag'  
You need a passphrase to unlock the secret key for  
user: "Scott Chacon <schacon@gee-mail.com>"  
1024-bit DSA key, ID F721C45A, created 2009-02-09
```

❑ ***git show v1.5***

```
$ git show v1.5  
tag v1.5  
Tagger: Scott Chacon <schacon@gee-mail.com>  
Date: Mon Feb 9 15:22:20 2009 -0800  
  
my signed 1.5 tag  
-----BEGIN PGP SIGNATURE-----  
Version: GnuPG v1.4.8 (Darwin)  
  
iEYEABECAAYFAkmQurIACgkQON3DxfchxFr5cACeIMN+ZxLKggJQf0QYiQBwgySN  
Ki0An2JeAVUCAiJ70x6ZEtK+NvZAJ82/  
=WryJ  
-----END PGP SIGNATURE-----  
commit 15027957951b64cf874c3557a0f3547bd83b3ff6  
Merge: 4a447f7... a6b4c97...  
Author: Scott Chacon <schacon@gee-mail.com>  
Date: Sun Feb 8 19:02:46 2009 -0800  
  
Merge branch 'experiment'
```



Легковажні мітки

☐ ***git tag v1.4-lw***

☐ ***git tag***

```
$ git tag v1.4-lw
$ git tag
v0.1
v1.3
v1.4
v1.4-lw
v1.5
```

☐ ***git show v1.4-lw***

```
$ git show v1.4-lw
commit 15027957951b64cf874c3557a0f3547bd83b3ff6
Merge: 4a447f7... a6b4c97...
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sun Feb 8 19:02:46 2009 -0800

Merge branch 'experiment'
```



Верифікація міток

❑ *git tag -v v1.4.2.1*

```
$ git tag -v v1.4.2.1
object 883653babd8ee7ea23e6a5c392bb739348b1eb61
type commit
tag v1.4.2.1
tagger Junio C Hamano <junkio@cox.net> 1158138501 -0700

GIT 1.4.2.1

Minor fixes since 1.4.2, including git-mv and git-http with alternates.
gpg: Signature made Wed Sep 13 02:08:25 2006 PDT using DSA key ID F3119B9A
gpg: Good signature from "Junio C Hamano <junkio@cox.net>"
gpg:          aka "[jpeg image of size 1513]"
Primary key fingerprint: 3565 2A26 2040 E066 C9A7 4A7D C0C6 D9A4 F311 9B9A
```

❑ *Ключ відсутній*

```
gpg: Signature made Wed Sep 13 02:08:25 2006 PDT using DSA key ID F3119B9A
gpg: Can't check signature: public key not found
error: could not verify the tag 'v1.4.2.1'
```



Пізнє оновлення міток

❑ *git log --pretty=oneline*

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'
a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support
0d52aaab4479697da7686c15f77a3d64d9165190 one more thing
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'
0b7434d86859cc7b8c3d5e1dddfed66ff742fcbb added a commit function
4682c3261057305bdd616e23b64b0857d832627b added a todo file
166ae0c4d3f420721acbb115cc33848dfcc2121a started write support
9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile
964f16d36dfccde844893cac5b347e7b3d44abbc commit the todo
8a5cbc430f1a9c3d00faaeffd07798508422908a updated readme
```

❑ *git tag -a v1.2 -m 'version 1.2' 9fceb02*

❑ *git tag*

```
$ git tag
v0.1
v1.2
v1.3
v1.4
v1.4-lw
v1.5

$ git show v1.2
tag v1.2
Tagger: Scott Chacon <schacon@gee-mail.com>
Date: Mon Feb 9 15:32:16 2009 -0800

version 1.2
commit 9fceb02d0ae598e95dc970b74767f19372d61af8
Author: Magnus Chacon <mchacon@gee-mail.com>
Date: Sun Apr 27 20:43:35 2008 -0700

    updated rakefile
...
```




Обмін мітками

❑ *git push origin v1.5*

```
$ git push origin v1.5
Counting objects: 50, done.
Compressing objects: 100% (38/38), done.
Writing objects: 100% (44/44), 4.56 KiB, done.
Total 44 (delta 18), reused 8 (delta 1)
To git@github.com:schacon/simplegit.git
* [new tag]          v1.5 -> v1.5
```

❑ *git push origin --tags*

```
$ git push origin --tags
Counting objects: 50, done.
Compressing objects: 100% (38/38), done.
Writing objects: 100% (44/44), 4.56 KiB, done.
Total 44 (delta 18), reused 8 (delta 1)
To git@github.com:schacon/simplegit.git
* [new tag]          v0.1 -> v0.1
* [new tag]          v1.2 -> v1.2
* [new tag]          v1.4 -> v1.4
* [new tag]          v1.4-lw -> v1.4-lw
* [new tag]          v1.5 -> v1.5
```



Псевдоніми у Git

- ☐ ***git config --global alias.co checkout***
- ☐ ***git config --global alias.br branch***
- ☐ ***git config --global alias.ci commit***
- ☐ ***git config --global alias.st status***

- ☐ ***git config --global alias.unstage 'reset HEAD --'***
- ☐ ***git unstage fileA***
- ☐ ***git reset HEAD fileA***

- ☐ ***git config --global alias.last 'log -1 HEAD'***
- ☐ ***git last***

```
$ git last
commit 66938dae3329c7aebe598c2246a8e6af90d04646
Author: Josh Goebel <dreamer3@example.com>
Date:   Tue Aug 26 19:48:51 2008 +0800

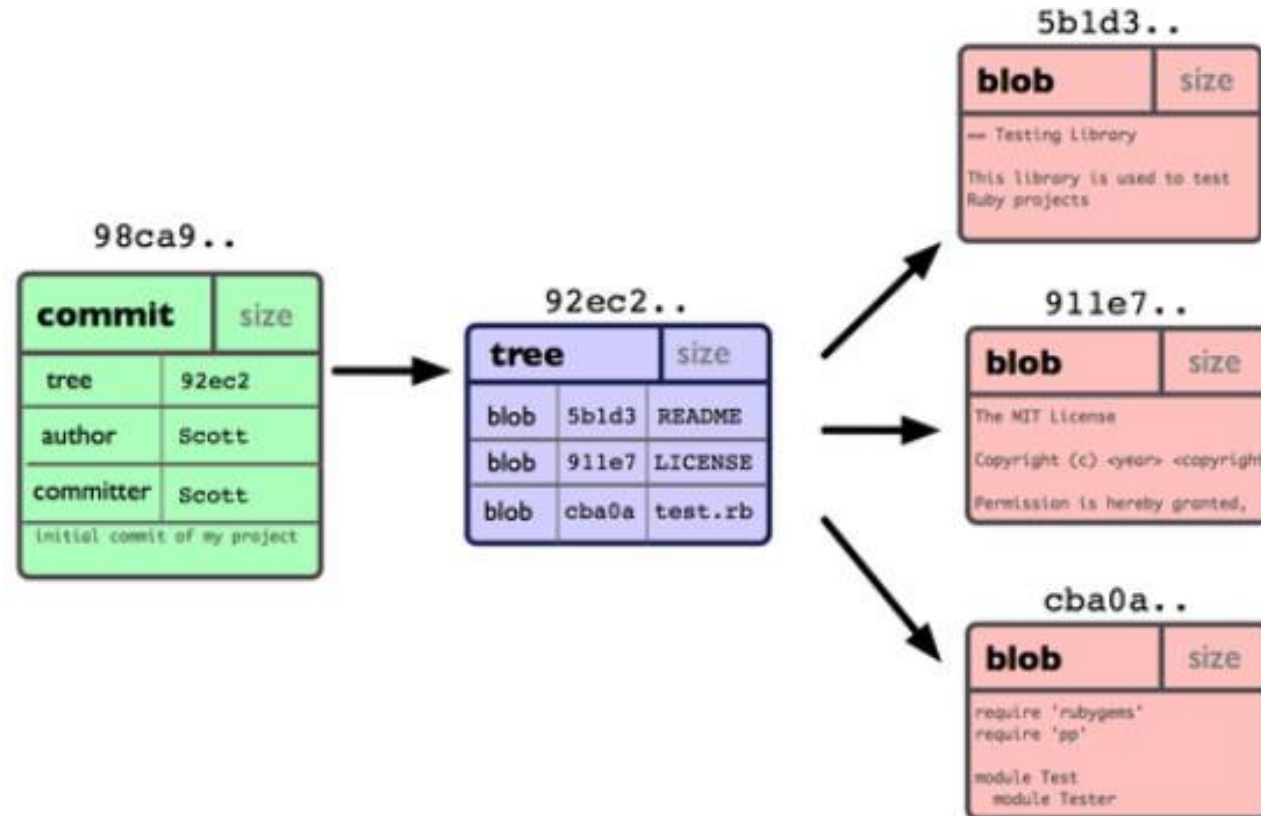
    test for current head

Signed-off-by: Scott Chacon <schacon@example.com>
```



Розгалуження у Git. Гілка

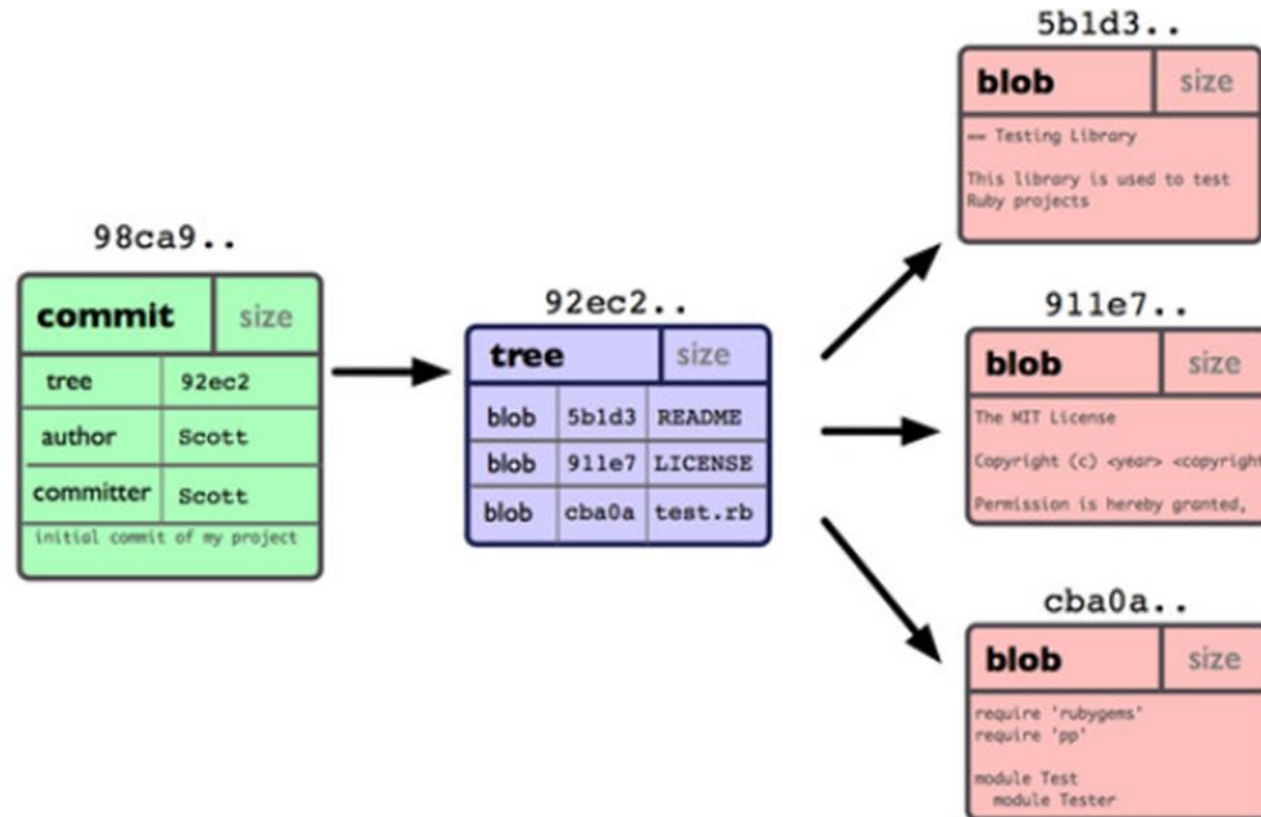
- ❑ `$ git add README test.rb LICENSE`
- ❑ `$ git commit -m 'initial commit of my project'`





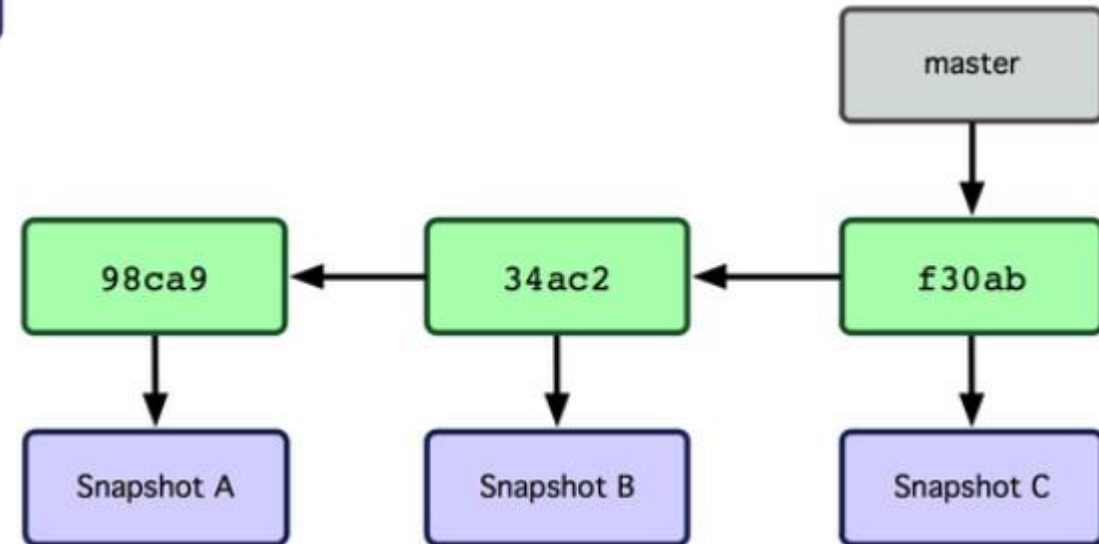
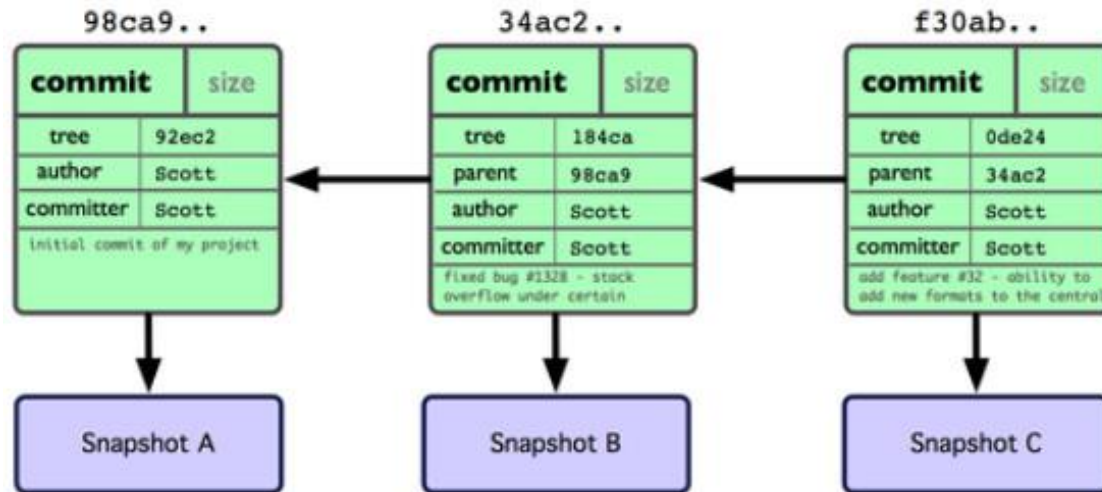
Розгалуження у Git. Гілка₁

- ❑ `$ git add README test.rb LICENSE`
- ❑ `$ git commit -m 'initial commit of my project'`





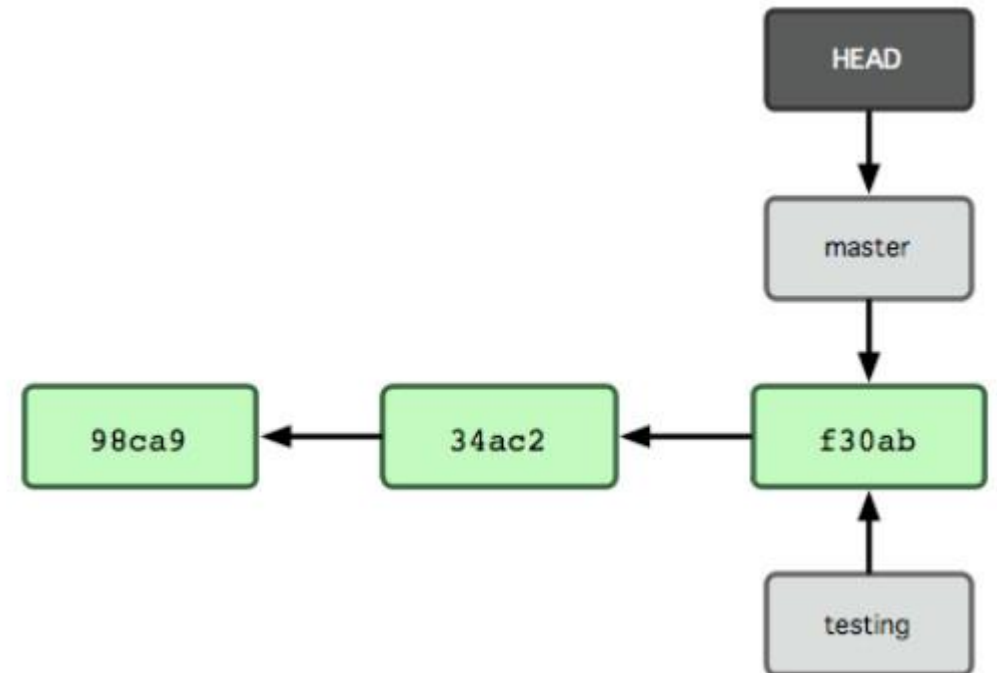
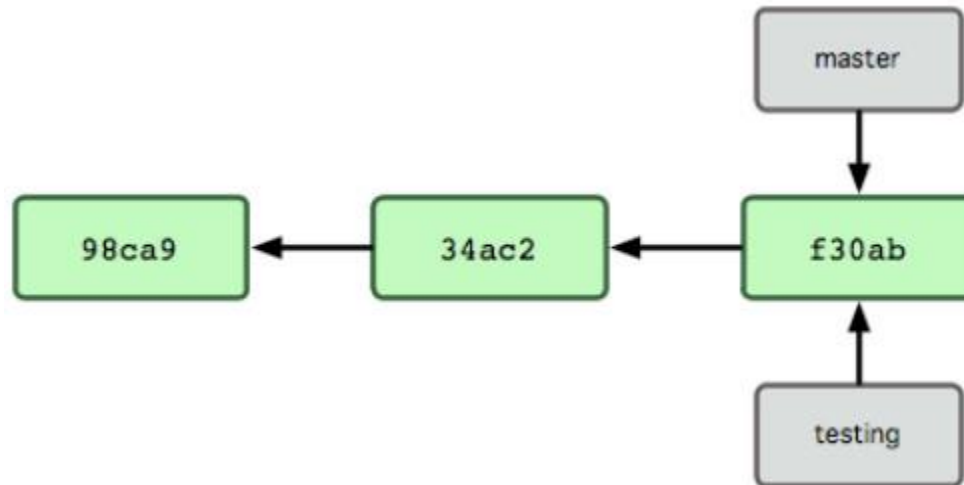
Розгалуження у Git. Гілка₂





Розгалуження у Git. Гілка₃

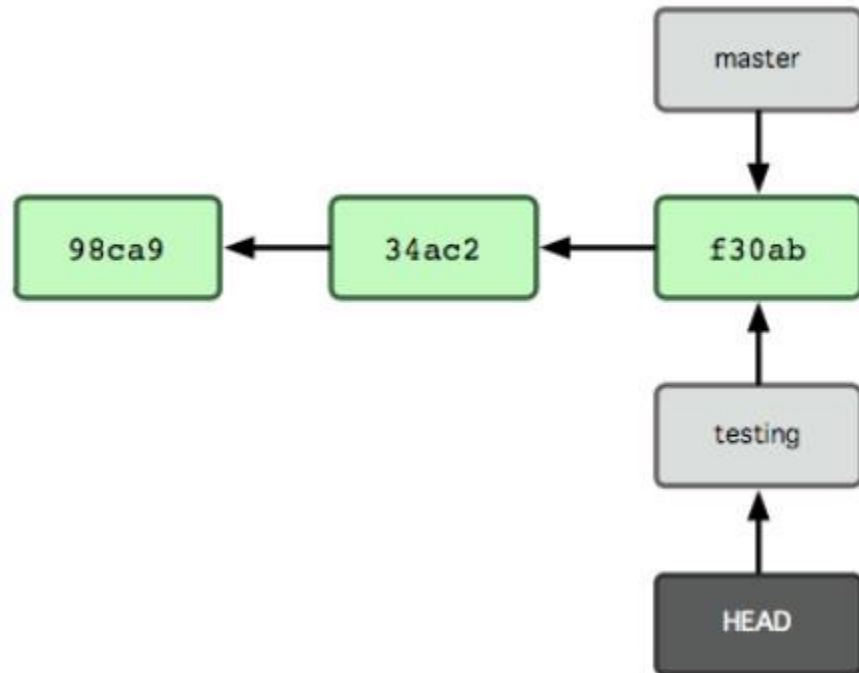
❑ *git branch testing*





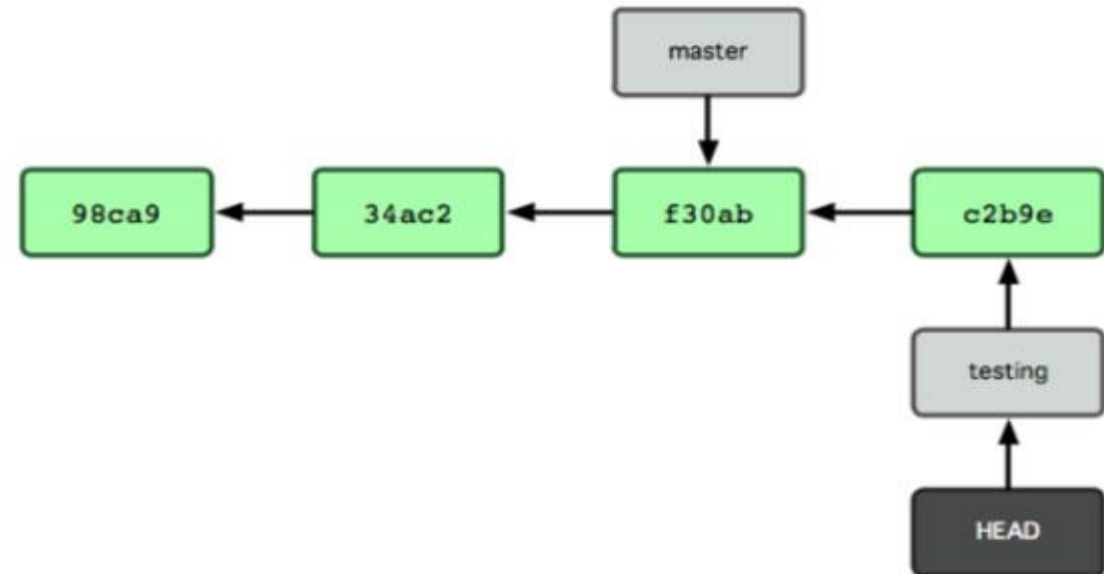
Розгалуження у Git. Гілка₄

❑ *git checkout testing*



❑ *vim test.rb*

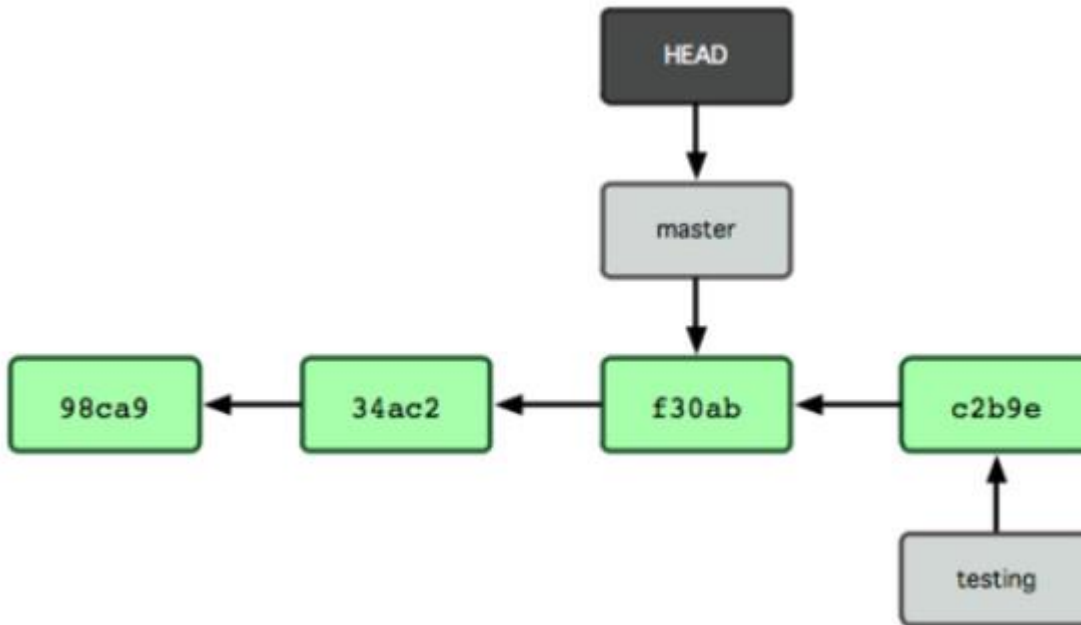
❑ *git commit -a -m 'made a change'*





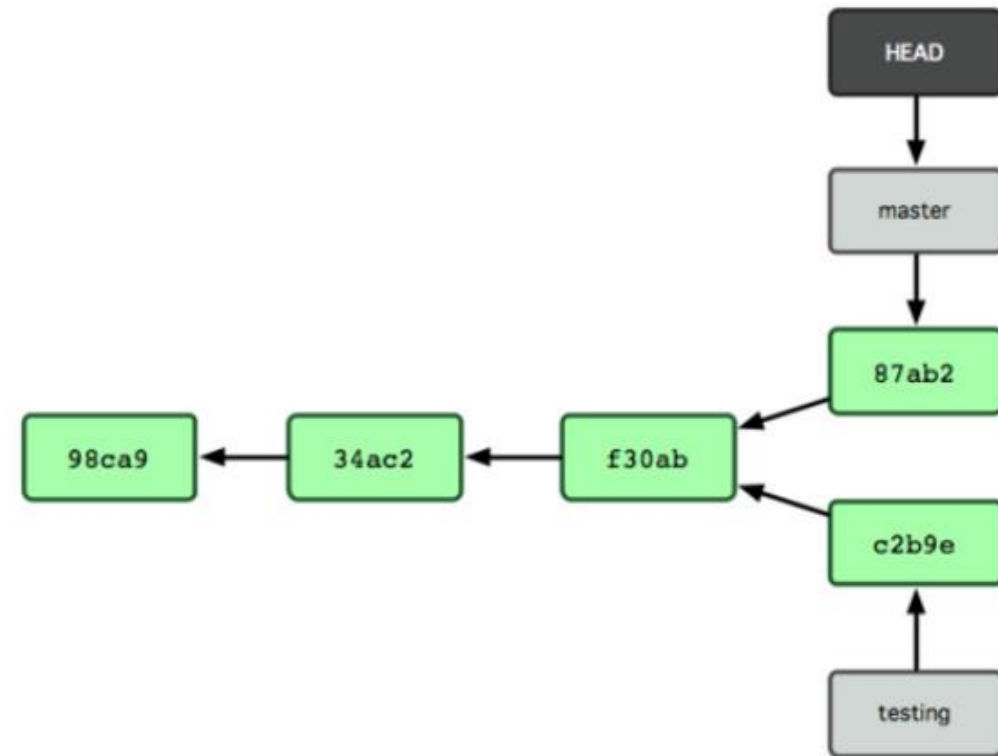
Розгалуження у Git. Гілка₅

❑ *git checkout master*



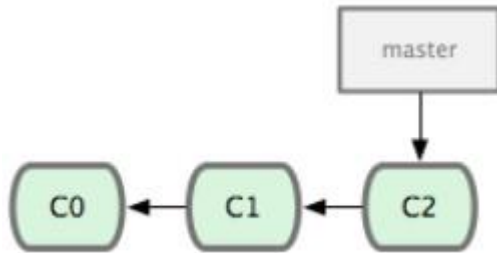
❑ *vim test.rb*

❑ *git commit -a -m 'made other change'*



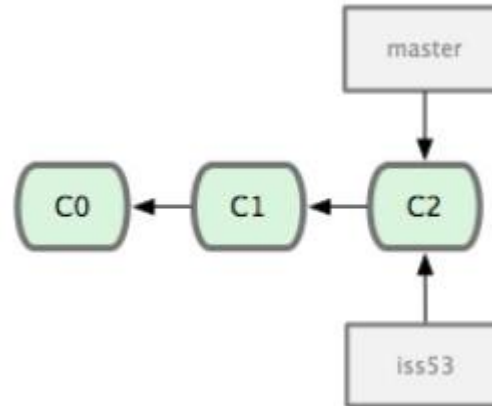


Основи розгалуження₁



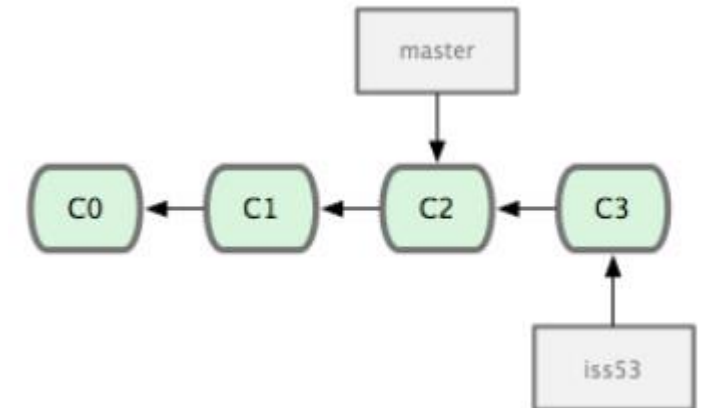
☐ ***git checkout -b iss53***

- `git branch iss53`
- `git checkout iss53`



☐ ***vim index.html***

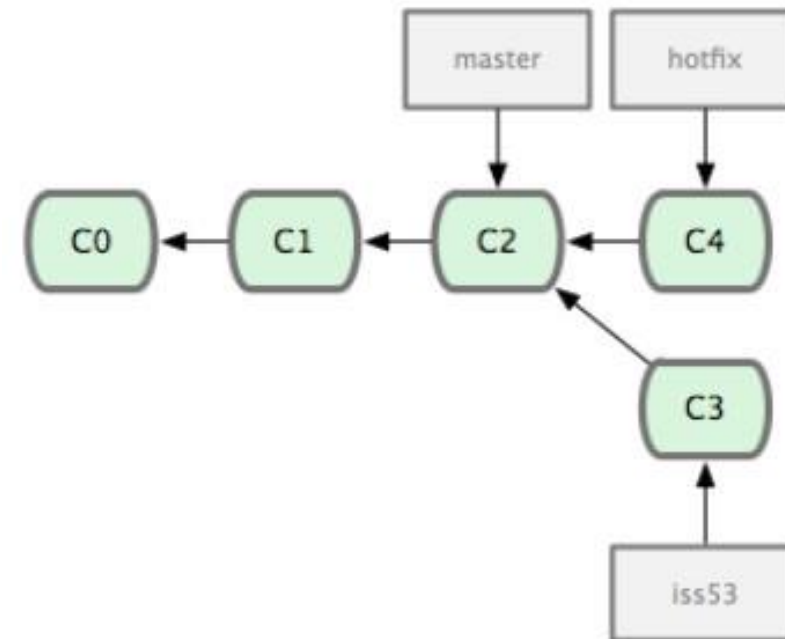
☐ ***\$ git commit -a -m 'added a new footer [issue 53]'***





Основи розгалуження₂

- ☐ *git checkout master*
- ☐ *git checkout -b hotfix*
- ☐ *vim index.html*
- ☐ *git commit -a -m 'fixed the broken email address'*
- ☐ *git checkout master*
- ☐ *git merge hotfix*



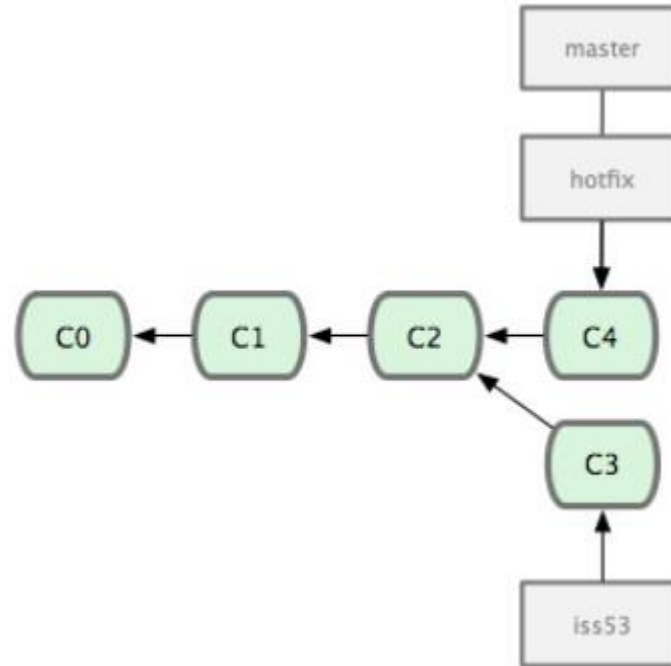
```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast forward
 README |    1 -
 1 files changed, 0 insertions(+), 1 deletions(-)
```



Основи розгалуження₃

☐ *git checkout master*

☐ *git merge hotfix*

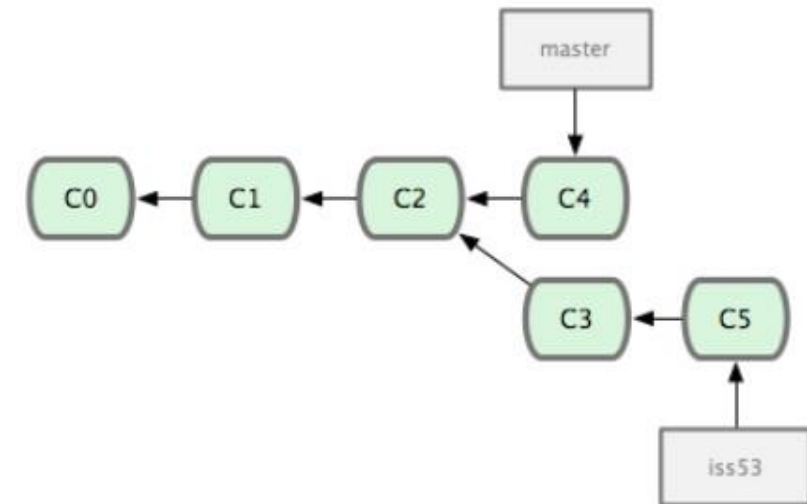


☐ *git branch -d hotfix*

☐ *git checkout iss53*

☐ *vim index.html*

☐ *git commit -a -m 'finished the new footer [issue 53]'*

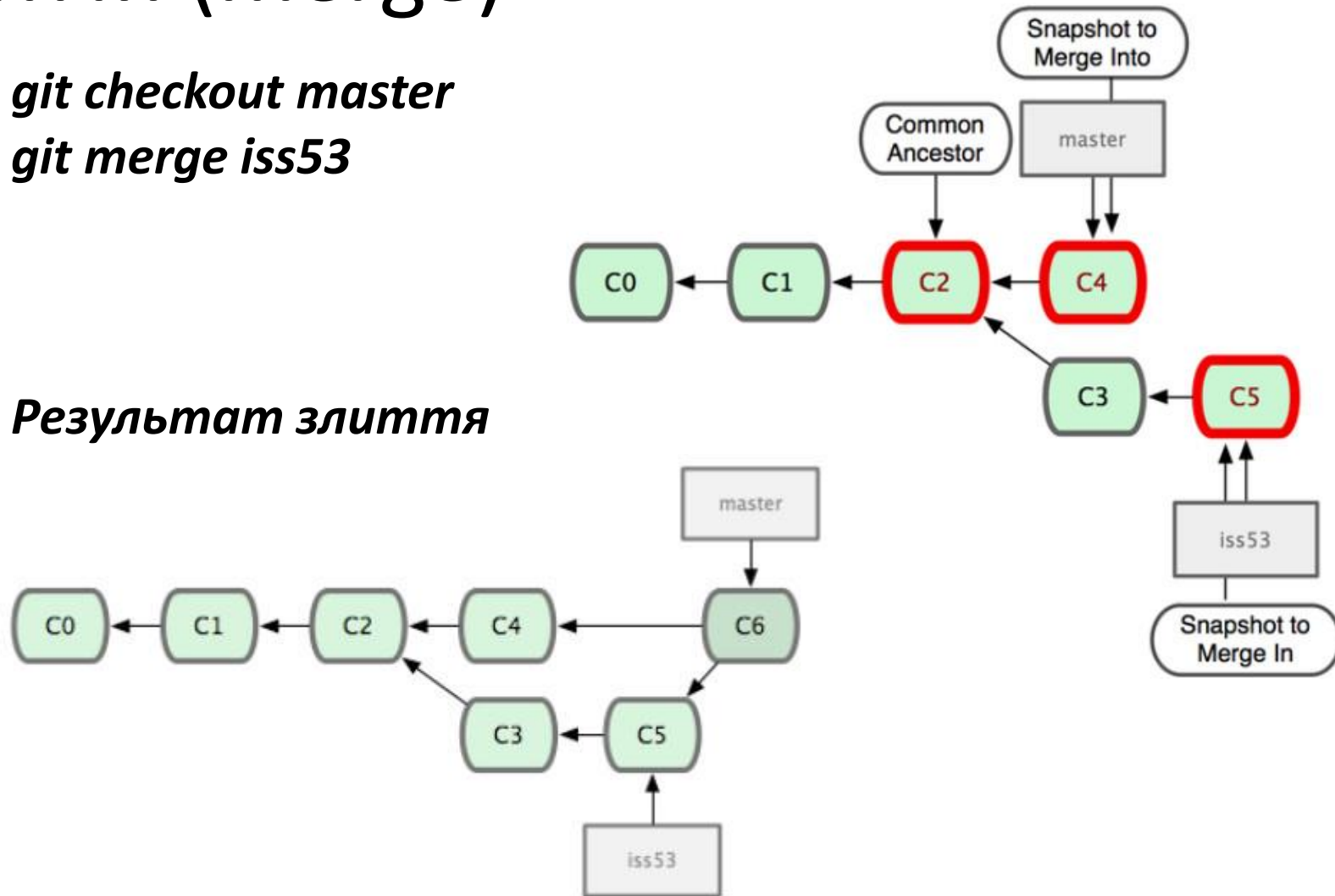




Злиття (merge)

- ☐ *git checkout master*
- ☐ *git merge iss53*

- ☐ *Результат злиття*





Конфлікти при злитті₁

❑ *git merge iss53*

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

❑ *git status*

```
[master*]$ git status
index.html: needs merge
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
```

```
#       unmerged:   index.html
#
```

```
<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>> iss53:index.html
```



Конфлікти при злитті₂

```
<div id="footer">
  please contact us at email.support@github.com
</div>
```

❑ *git mergetool*

```
$ git mergetool
merge tool candidates: kdiff3 tkdiff xxdiff meld gvimdiff opendiff emerge vimdiff
Merging the files: index.html

Normal merge conflict for 'index.html':
  {local}: modified
  {remote}: modified
Hit return to start merge resolution tool (opendiff):
```



Конфлікти при злитті₃

❑ *git status*

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   index.html
#
```

```
Merge branch 'iss53'

Conflicts:
  index.html
#
# It looks like you may be committing a MERGE.
# If this is not correct, please remove the file
# .git/MERGE_HEAD
# and try again.
#
```



Управління гілками₁

❑ *git branch*

```
$ git branch  
iss53  
* master  
testing
```

❑ *git branch -v*

```
$ git branch -v  
iss53 93b412c fix javascript issue  
* master 7a98805 Merge branch 'iss53'  
testing 782fd34 add scott to the author list in the readmes
```

❑ *git branch --merged*

```
$ git branch --merged  
iss53  
* master
```



Управління гілками₂

☐ *git branch --no-merged*

```
$ git branch --no-merged  
testing
```

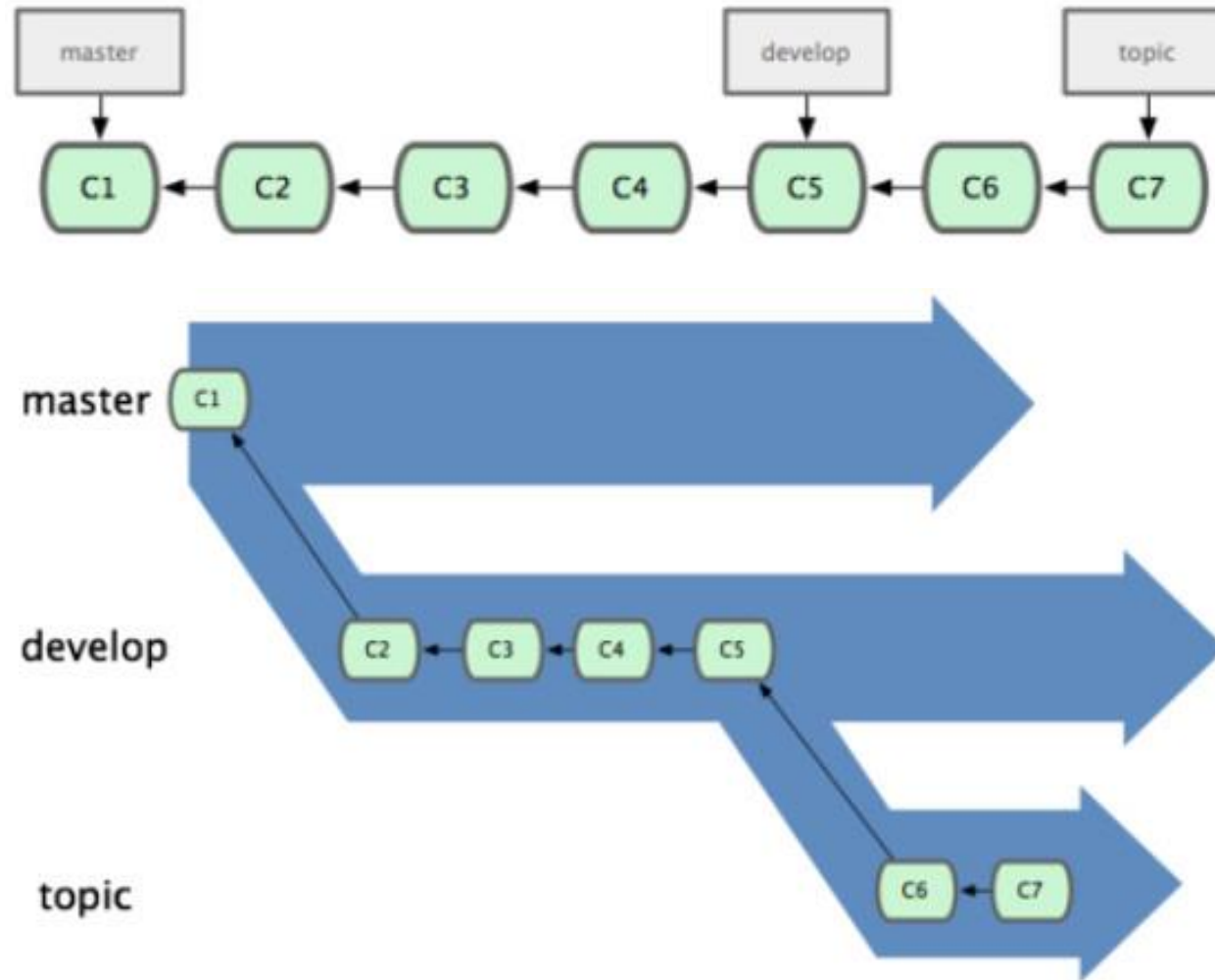
☐ *git branch -d testing*

```
$ git branch -d testing  
error: The branch 'testing' is not an ancestor of your current HEAD.  
If you are sure you want to delete it, run 'git branch -D testing'.
```

☐ *git branch -D testing*

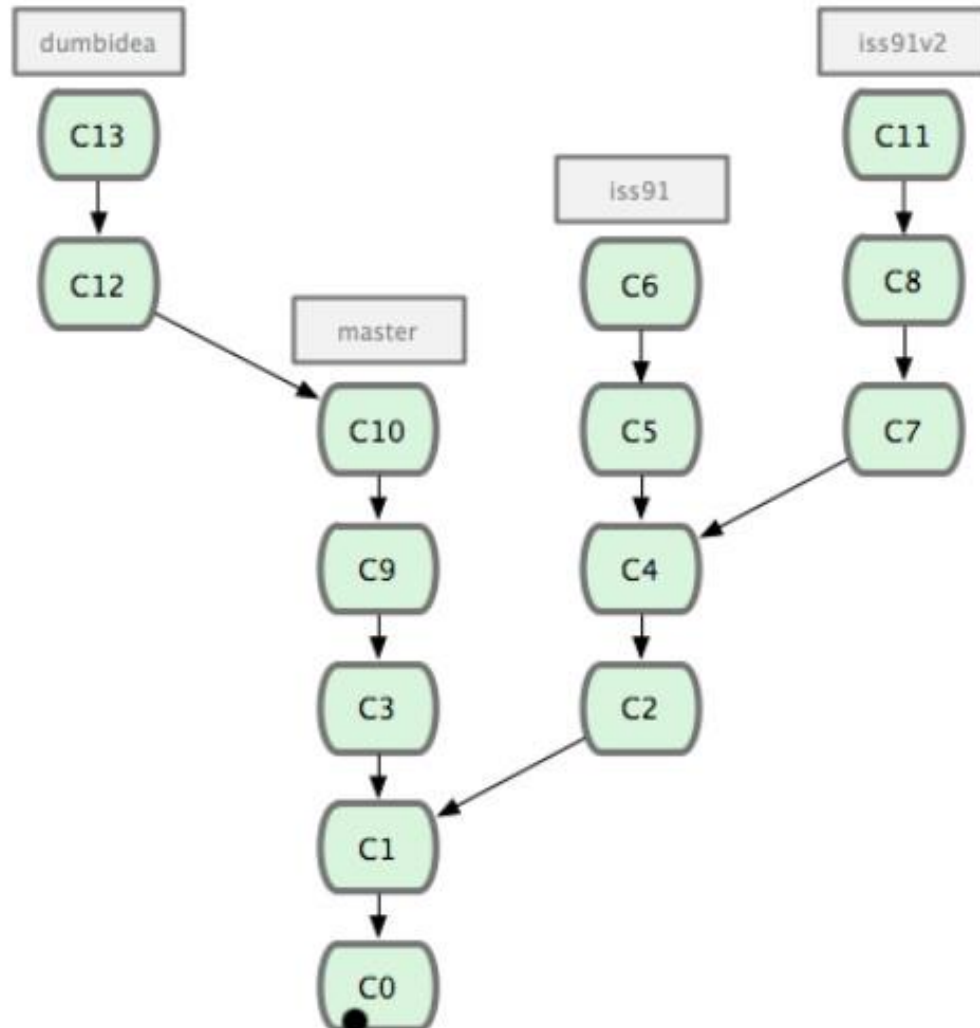


Довгоживучі гілки



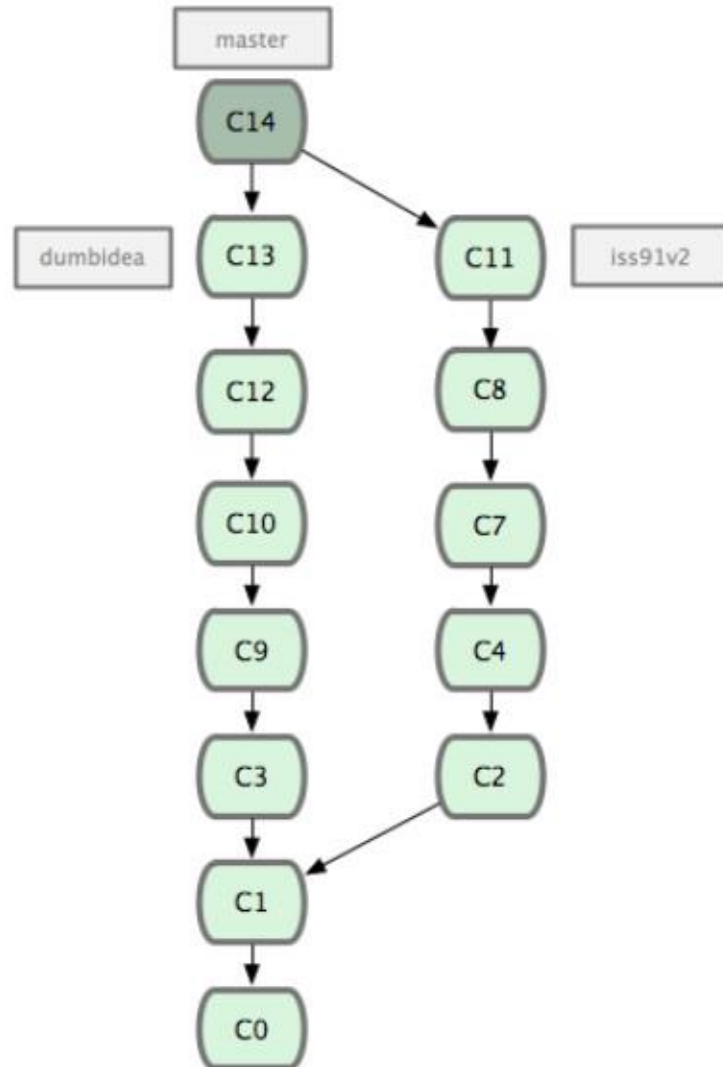


Тематичні гілки₁



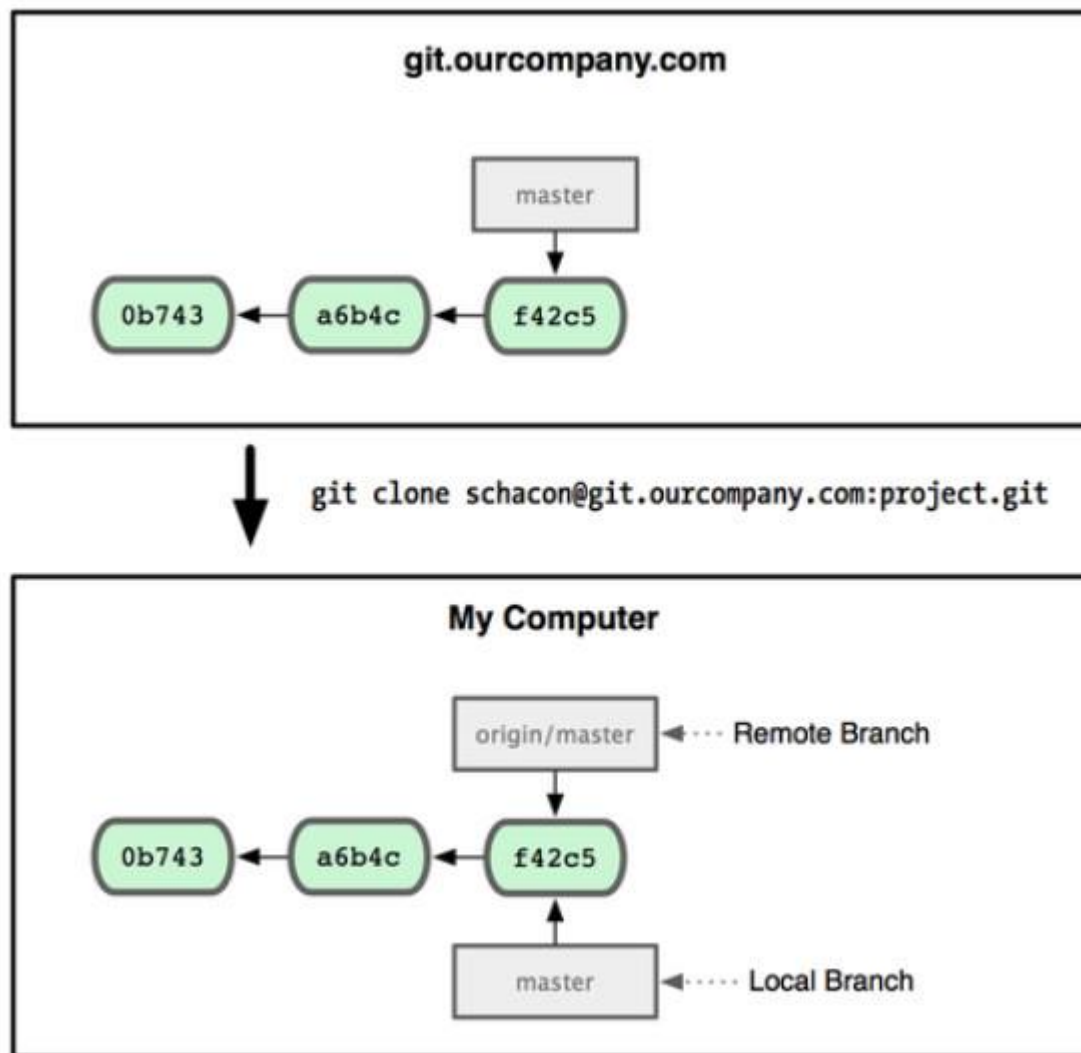


Тематичні гілки₂



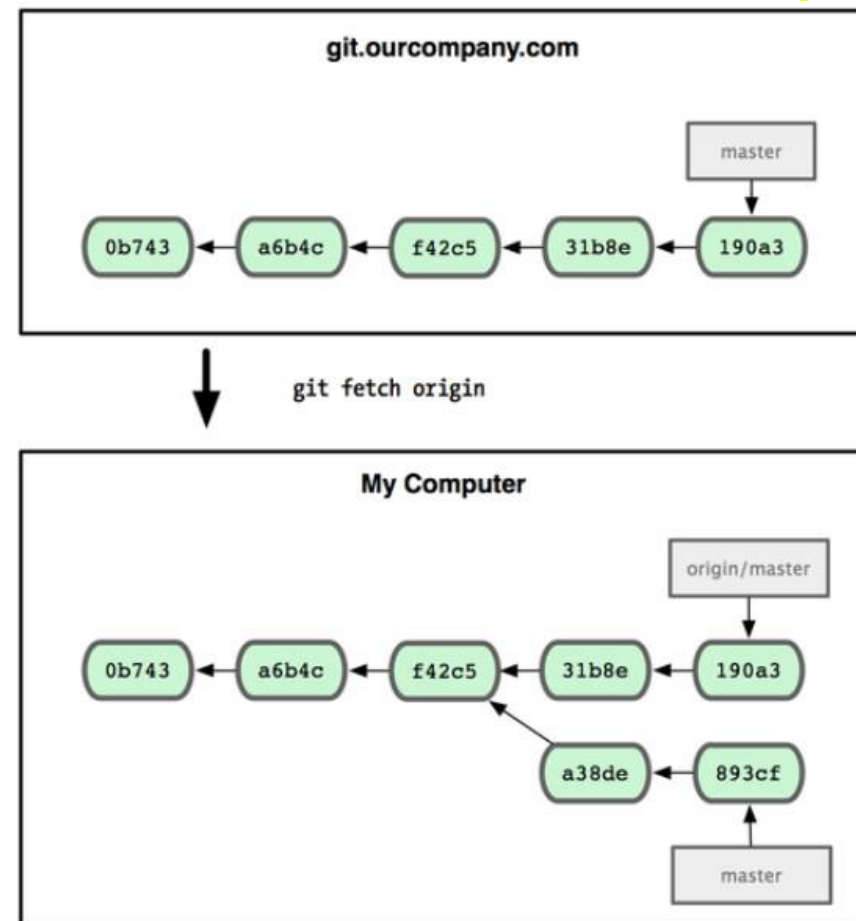
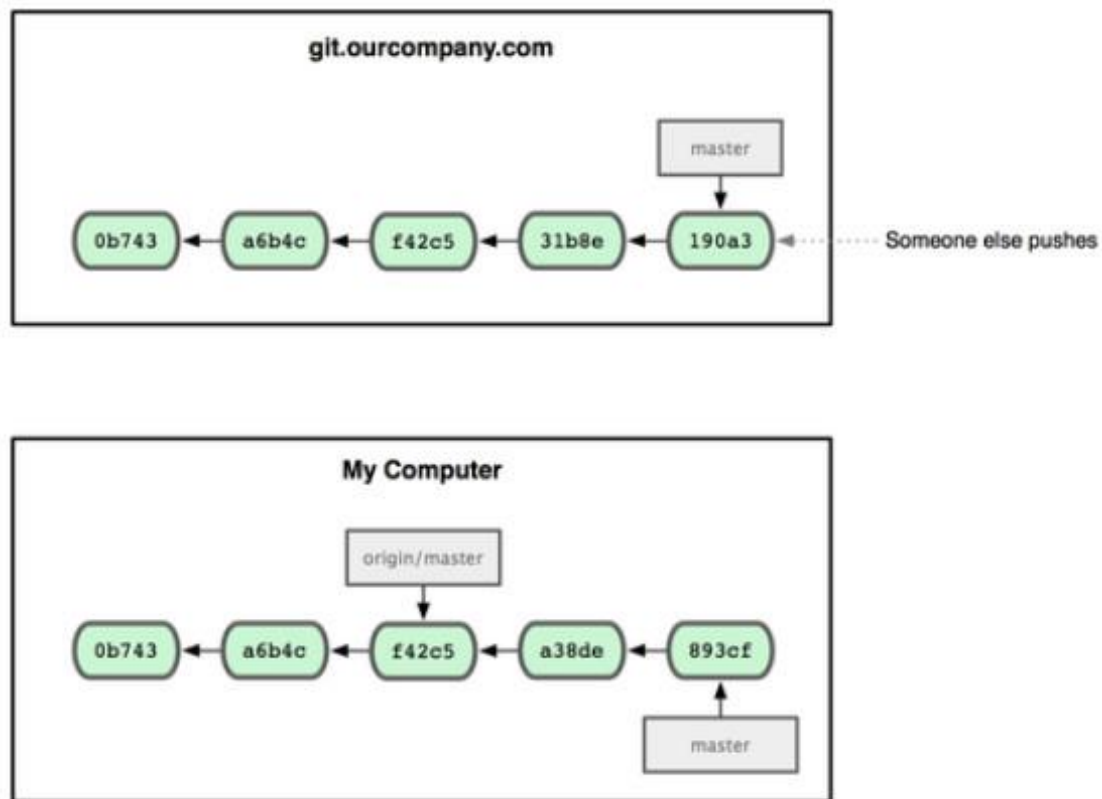


Віддалені гілки₁



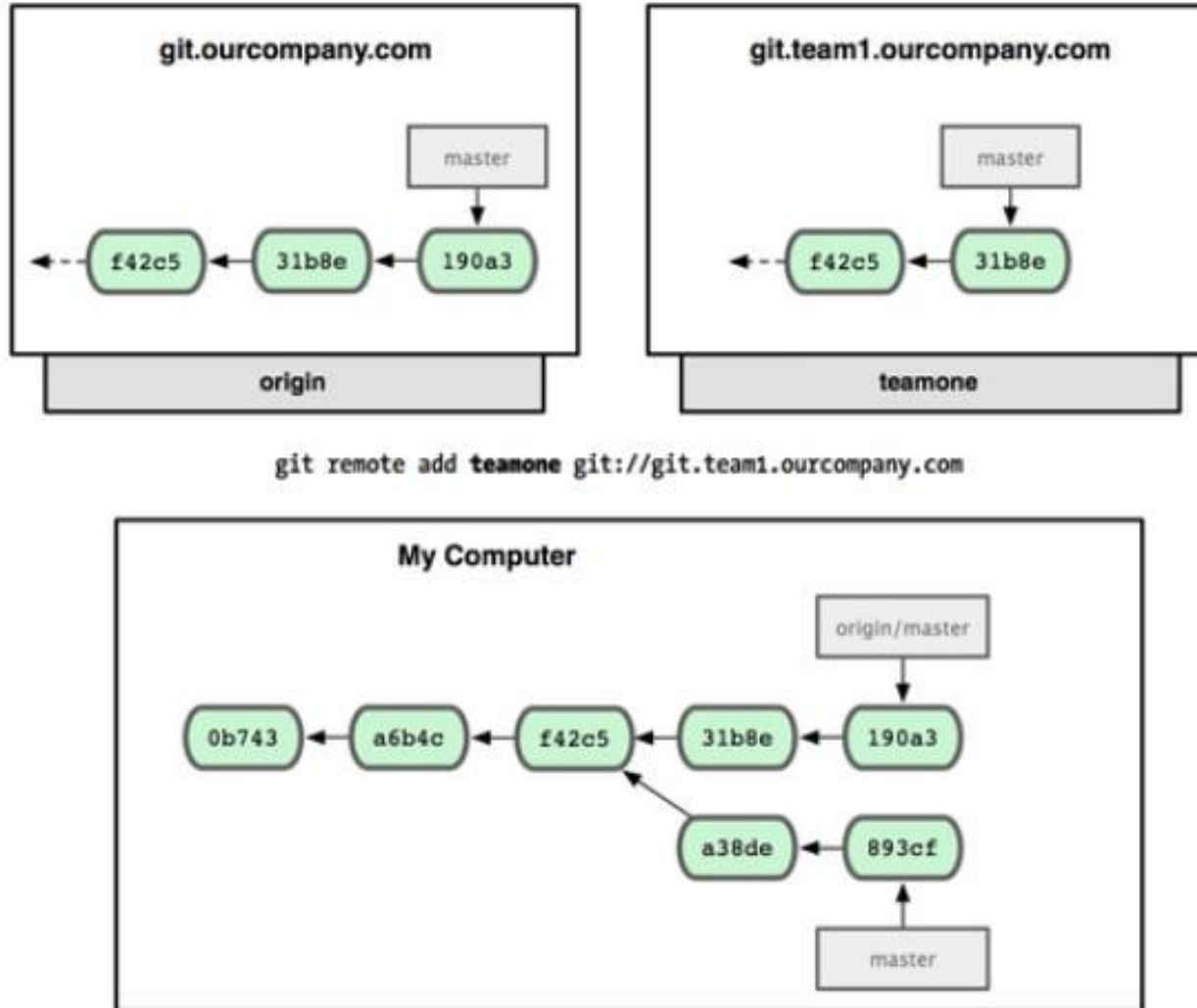


Віддалені гілки₂





Віддалені гілки₃





Надсилання змін

❑ ***git push origin serverfix***

```
$ git push origin serverfix
Counting objects: 20, done.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.74 KiB, done.
Total 15 (delta 5), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new branch]      serverfix -> serverfix
```

❑ ***git fetch origin***

```
$ git fetch origin
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15 (delta 5), reused 0 (delta 0)
Unpacking objects: 100% (15/15), done.
From git@github.com:schacon/simplegit
 * [new branch]      serverfix -> origin/serverfix
```

❑ ***git checkout -b serverfix origin/serverfix***

```
$ git checkout -b serverfix origin/serverfix
Branch serverfix set up to track remote branch refs/remotes/origin/serverfix.
Switched to a new branch "serverfix"
```



Відстеження гілок

☐ *git checkout --track origin/serverfix*

```
$ git checkout --track origin/serverfix  
Branch serverfix set up to track remote branch refs/remotes/origin/serverfix.  
Switched to a new branch "serverfix"
```

☐ *git checkout -b sf origin/serverfix*

```
$ git checkout -b sf origin/serverfix  
Branch sf set up to track remote branch refs/remotes/origin/serverfix.  
Switched to a new branch "sf"
```



Видалення гілок на віддаленому сервері

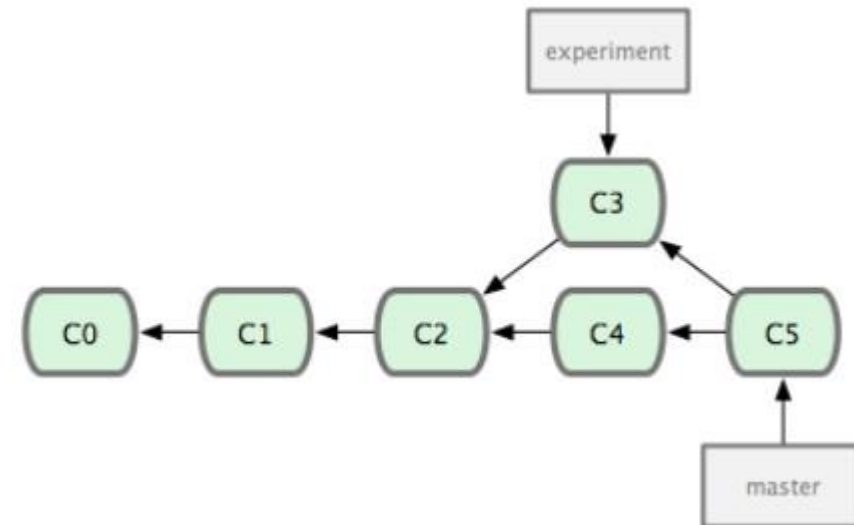
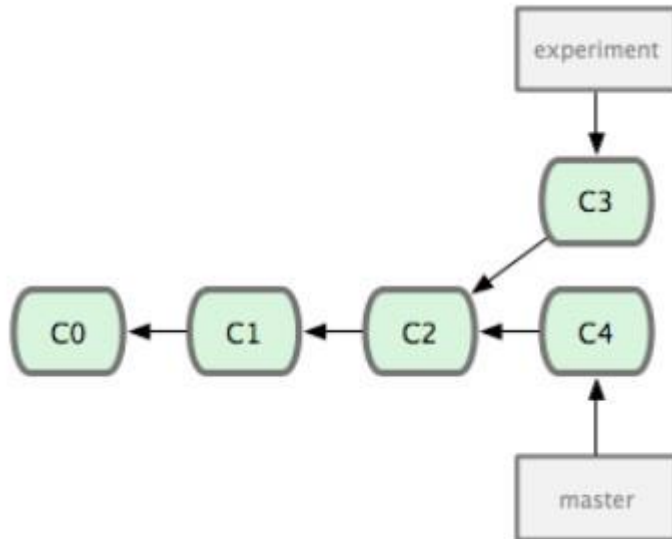
❑ *git push origin :serverfix*

```
$ git push origin :serverfix
To git@github.com:schacon/simplegit.git
- [deleted]          serverfix
```

123



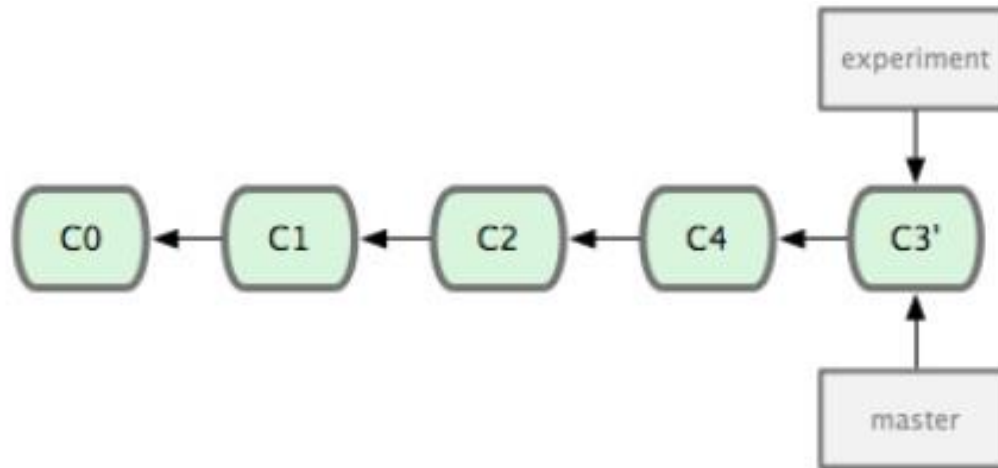
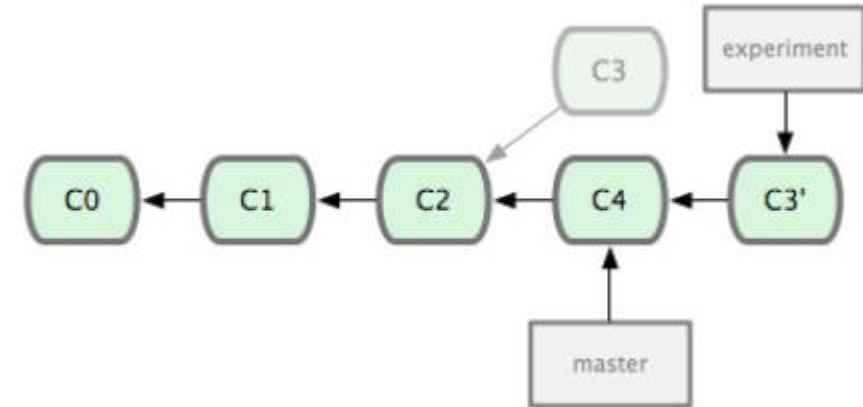
Переміщення гілок₁





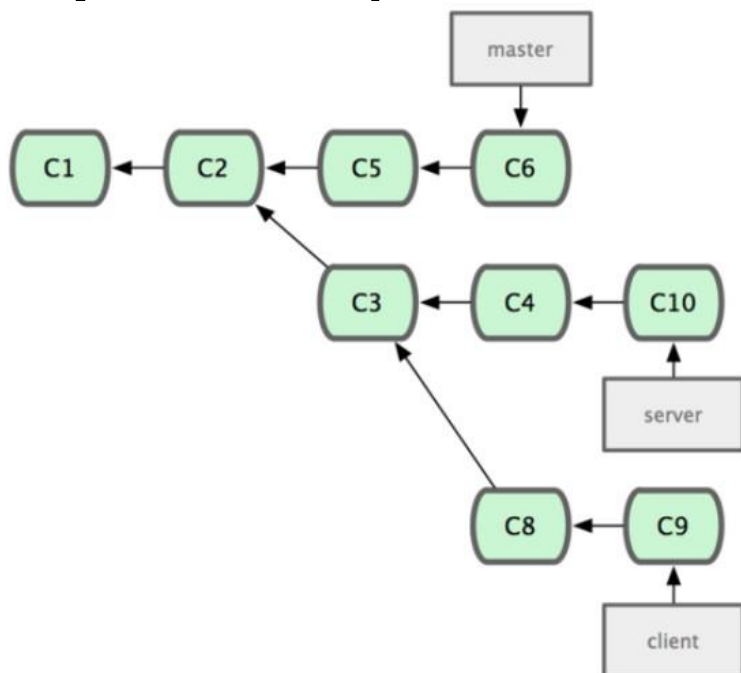
Переміщення гілок₂

- ☐ *git checkout experiment*
- ☐ *git rebase master*

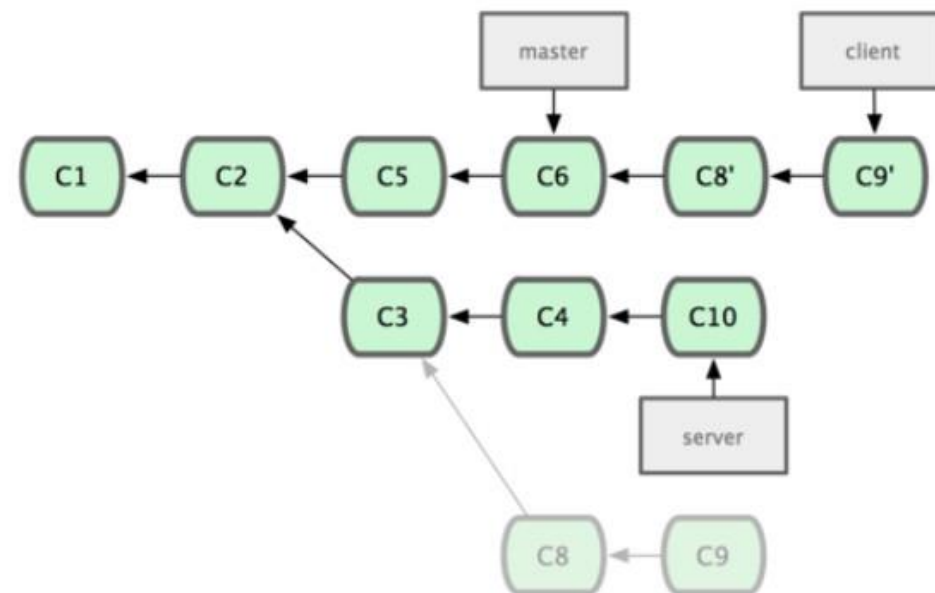




Переміщення гілок₃



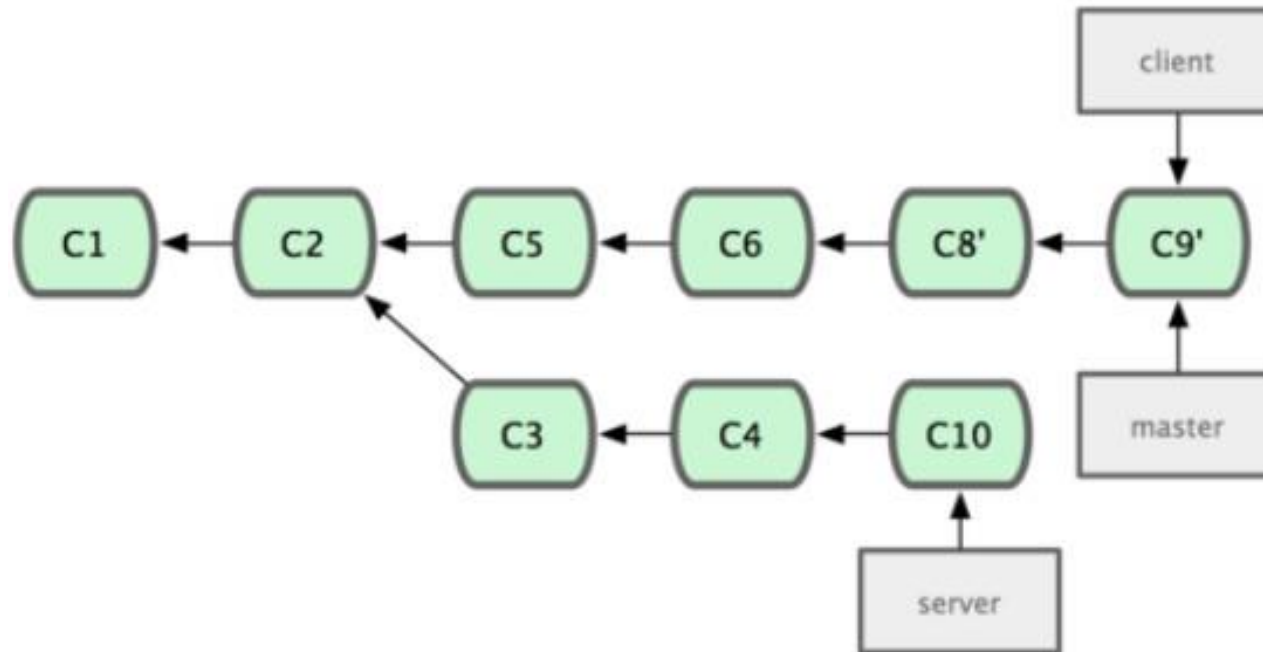
❑ *git rebase --onto master server client*





Переміщення гілок₄

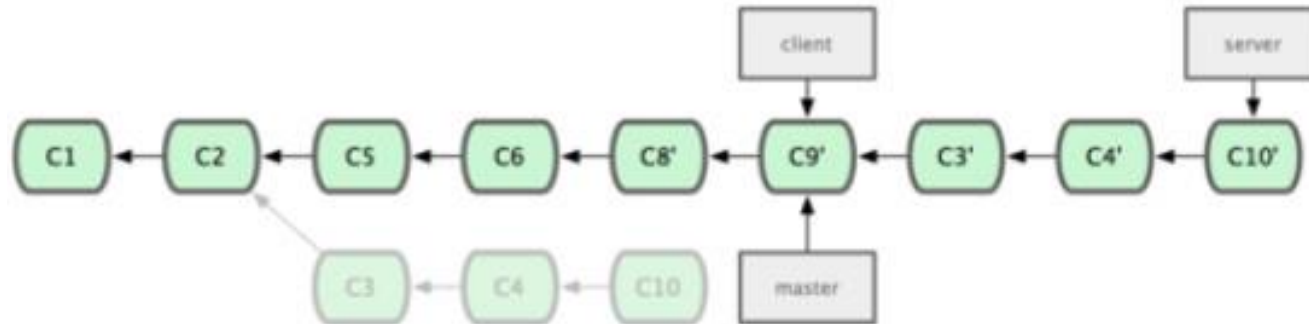
- ☐ *git checkout master*
- ☐ *git merge client*





Переміщення гілок₅

☐ *git rebase master server*

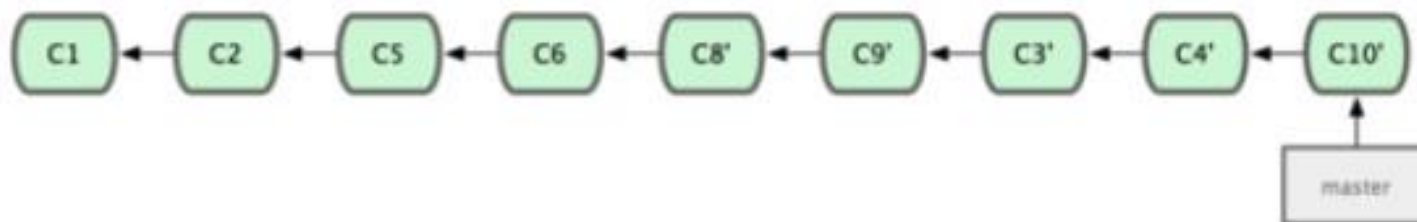


☐ *git checkout master*

☐ *git merge server*

☐ *git branch -d client*

☐ *git branch -d server*



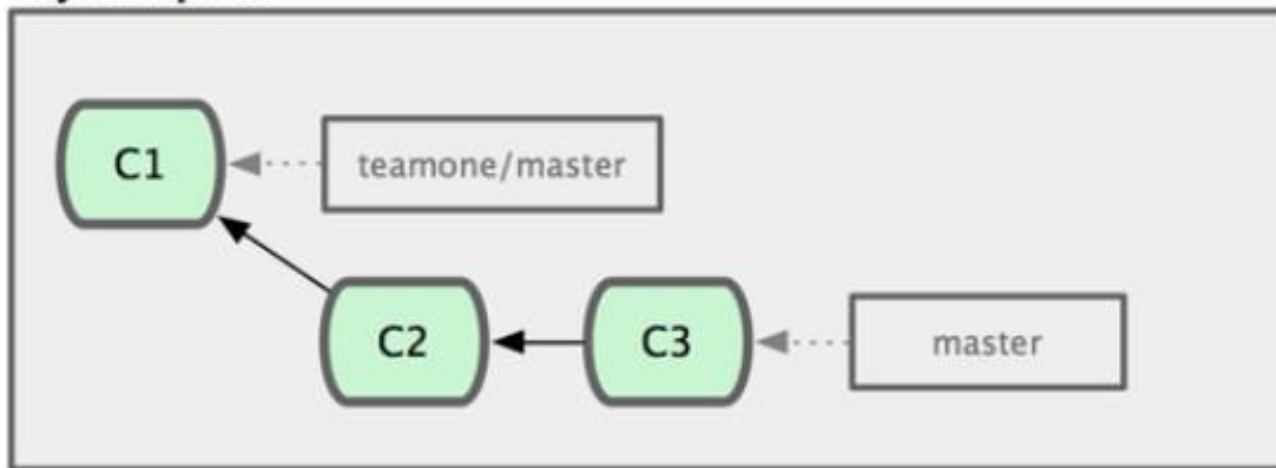


Можливі ризики переміщення₁

git.team1.ourcompany.com



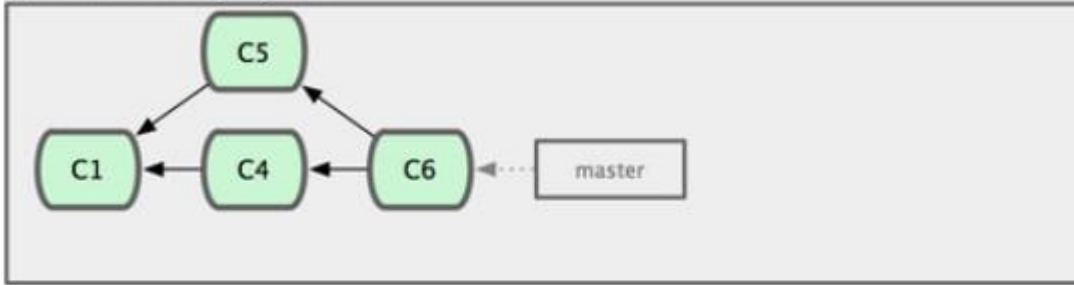
My Computer



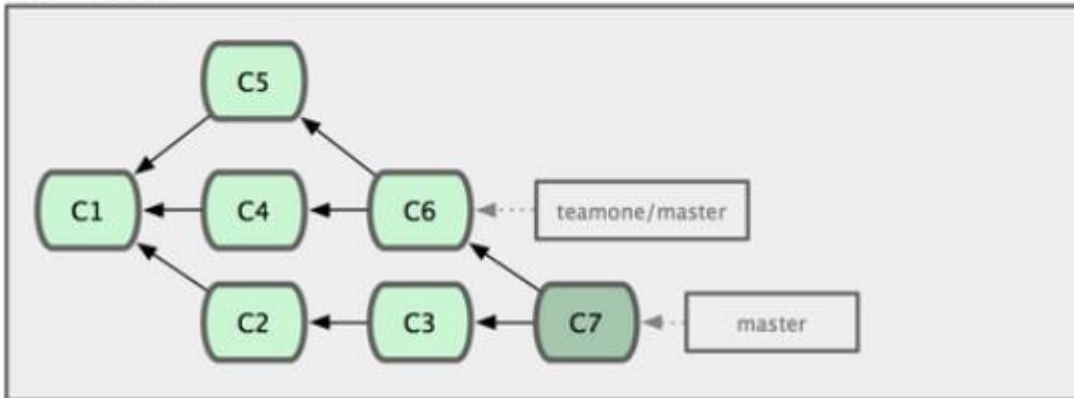


Можливі ризики переміщення₂

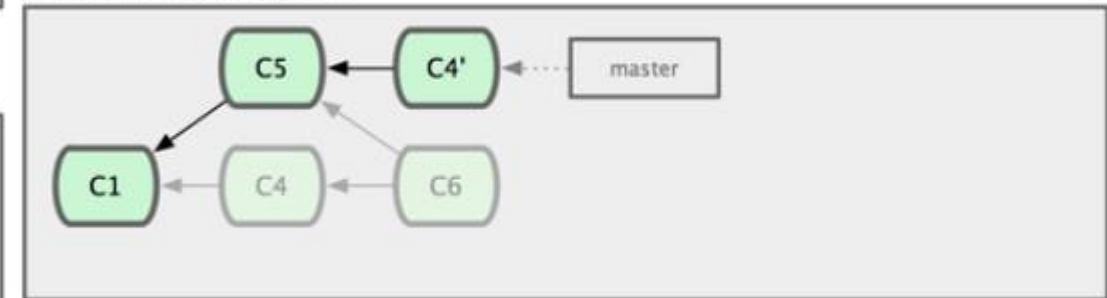
git.team1.ourcompany.com



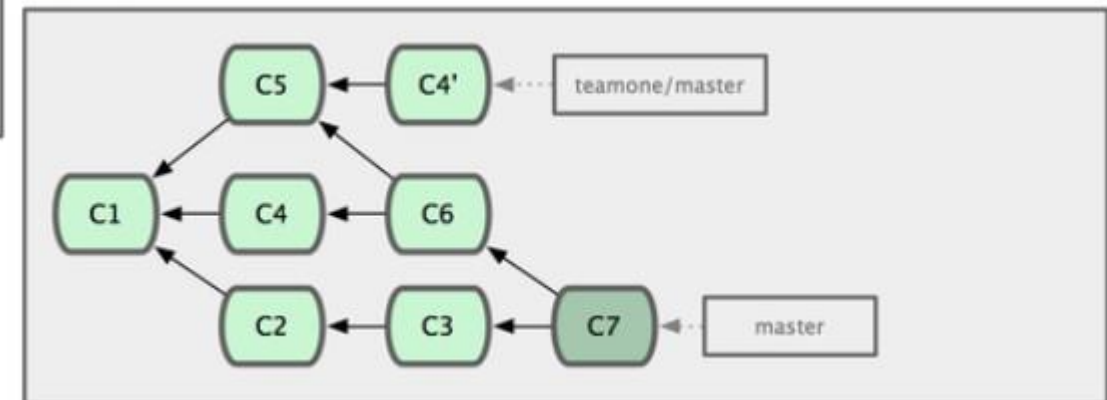
My Computer



git.team1.ourcompany.com



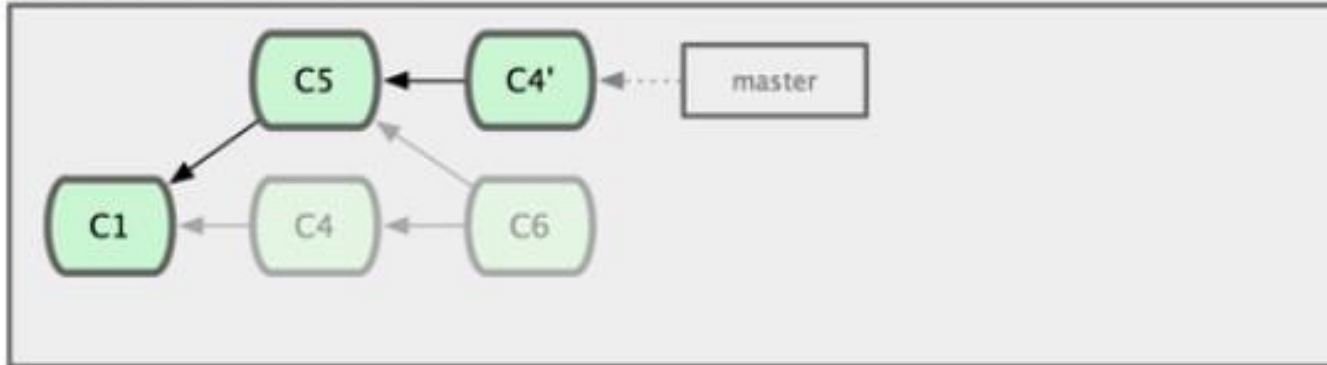
My Computer



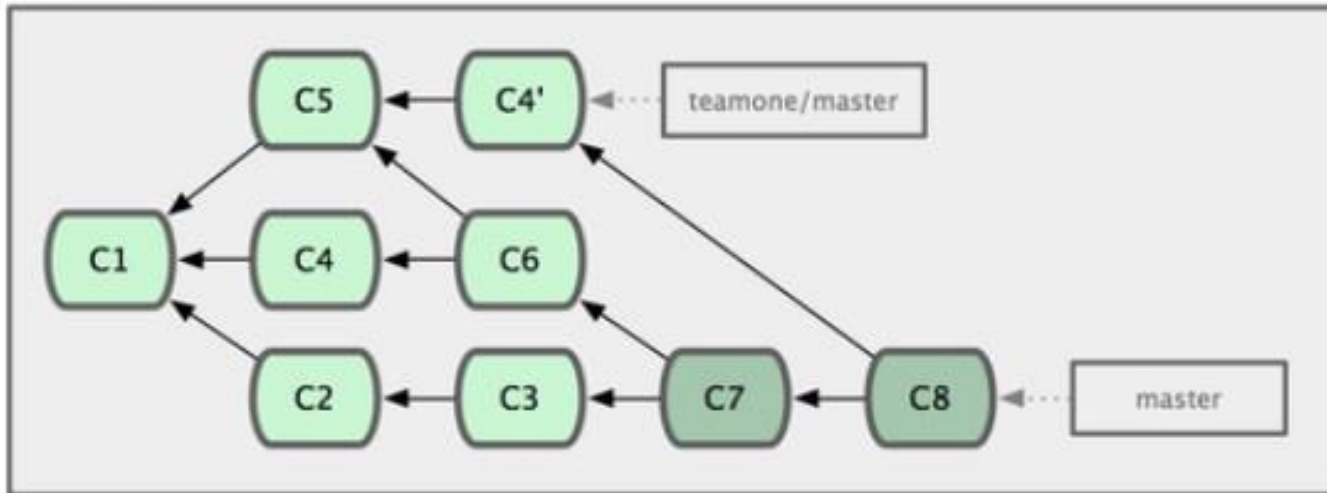


Можливі ризики переміщення₃

git.team1.ourcompany.com



My Computer



Донецький національний технічний університет



Тема 3. Bash та Python для DevOps. Лекція 4. Основи BASH

Предмет: Технології DevOPS

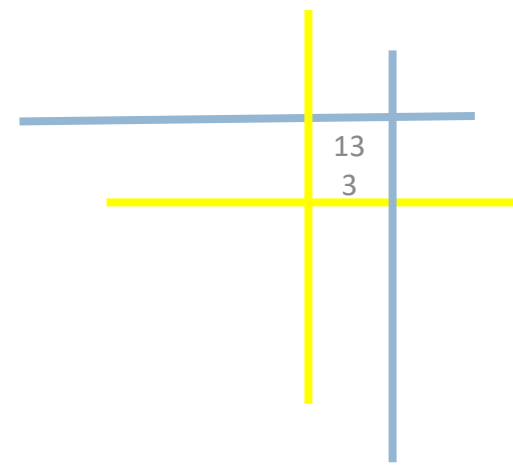
Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання



- ☐ Мета лекції
- ☐ Робота в BASH
- ☐ Основні елементи мови BASH



Мета лекції

134

☐ Розглянути основні елементи мови BASH



Скрипт (сценарій)

❑ **Сценарій** — це файл, який містить команди оболонки

- Структури даних: змінні
- Управляючі структури: послідовність, рішення, цикл

❑ Shebang рядок для скрипта bash:

❑ **`#!/bin/bash`**

❑ **Запуск сценарію:**

- Зробити файл виконуваним: **`% chmod +x script`**
- Викликати: **`% ./script`**



Основні елементи мови

□ Введення

- запит користувача
- аргументи командного рядка

□ Розгалуження

- `if-then-else`
- `case`

□ Цикл

- `do-while`, `repeat-until`
- `for`
- `select`

□ Функції

□ Traps



Користувацьке введення

❑ оболонка дозволяє запитувати введення користувача:

- `read varname [more vars]`
- `або`
- `read -p "prompt" varname [more vars]`

❑ слова, введені користувачем, призначаються змінним `varname` та `"more vars"`

❑ остання змінна отримує решту рядка введення



Приклад

138

```
#!/bin/sh  
read -p "enter your name: " first last  
echo "First name: $first"    echo "Last name:  
$last"
```



Спеціальні змінні оболонки

Змінна	Значення
\$0	Назва скрипта, який виконується
\$1–\$9	Позиційний параметр 1-9
\$#	Кількість позиційних параметрів
\$*	Всі позиційні параметри, “\$*” – один рядок
\$@	Всі позиційні параметри, “\$@” – множина рядків
\$?	Повертає статус останньої виконаної команди
\$\$	Id поточного процесу



Приклади з командним рядком

```
% set tim bill ann fred  
      $1 $2 $3 $4
```

```
% echo $*  
tim bill ann fred
```

```
% echo $#  
4
```

```
% echo $1  
tim
```

```
% echo $3 $4  
ann fred
```




Управляючі структури BASH

☐ *if-then-else*

☐ *case*

☐ *цикли*

- *for*

- *while*

- *until*

- *select*



Твердження IF

```
if command  
then  
    statements  
fi
```

- ❑ *statements* виконується тільки, якщо *command* успішна, тобто повертає статус «0»



Команда TEST

Синтаксис:

```
test expression  
[ expression]
```

□ оцінює 'expression' та повертає true або false

Приклад:

```
if test -w "$1"  
then  
    echo "file $1 is writeable"  
fi
```



Просте твердження IF

```
if [ condition ]; then  
    statements  
fi
```

□ виконує `statements` при умові, якщо **`condition`** є `true`



Твердження IF-THEN-ELSE

```
if [ condition ]; then  
    statements-1  
else  
    statements-2  
fi
```

- ☐ statements-1 виконується, якщо condition є true
- ☐ statements-2, якщо condition є false



Твердження IF...

```
if [ condition ]; then
    statements
elif [ condition ]; then
    statement
else
    statements
fi
```

- ❑ твердження `elif` є скороченням “else if” та є частиною твердження `if`, а тому окремо використовуватися не може



Оператори порівняння

Значення	Числа	Рядки
Більше	-gt	
Більше або дорівнює	-ge	
Менше	-lt	
Менше або дорівнює	-le	
Дорівнює	-eg	
Не дорівнює	-ne	!=
str1 менше ніж str2		str1 < str2
str1 більше ніж str2		str1 > str2
Довжина рядка більше 0		-n str
Довжина рядка 0		-z str



Складні логічні вирази

!

НІ

&&

I

||

АБО

I, АБО

повинні бути в

[[

]]



Приклад використання оператора !

```
#!/bin/bash
```

```
read -p "Введіть кількість років роботи: " Years
if [ ! "$Years" -lt 35 ]; then
    echo " Можна йти на пенсію."
else
    echo " Потрібно 35+ років для виходу на пенсію"
fi
```



Приклад використання оператора & &

```
#!/bin/bash
```

```
Bonus=500
```

```
read -p " Введіть статус: " Status
```

```
read -p " Введіть зміну: " Shift
```

```
if [[ "$Status" = "Н" && "$Shift" = 3 ]]
```

```
then
```

```
    echo " зміна $Shift отримує \$$Bonus бонус"
```

```
else
```

```
    echo " тільки почасовики в "
```

```
    echo " зміні 3 отримують бонус "
```

```
fi
```



Приклад використання оператора | |

```
#!/bin/bash
```

```
read -p " Введіть кількість оброблених дзвінків:" CHandle
read -p " Введіть кількість завершених дзвінків: " CClose
if [[ "$CHandle" -gt 150 || "$CClose" -gt 50 ]]
then
    echo " Ви маєте право на бонус"
else
    echo " Ви отримуєте бонус якщо"
    echo " к-ть оброблених дзвінків більша 150 або"
    echo " к-ть завершених дзвінків більша 50 "
fi
```



Тестування файлів

Значення

-d file	True якщо 'file' є папкою
-f file	True якщо 'file' є файлом
-r file	True якщо 'file' є читабельним
-w file	True якщо 'file' є доступним для запису
-x file	True якщо 'file' є виконуваним
-s file	True якщо розмір 'file' є більшим за 0



Приклад тестування файлів₁

```
#!/bin/bash
echo "Введіть назву файлу: "
read filename
if [ ! -r "$filename" ]
then
    echo "Файл не доступний для читання"
    exit 1
fi
```



Приклад тестування файлів₂

```
#!/bin/bash
if [ $# -lt 1 ];
then
    echo "Використання: filetest filename"
    exit 1
fi

if [[ ! -f "$1" || ! -r "$1" || ! -w "$1" ]]
then
    echo "Файл $1 недоступний"
    exit 1
fi
```



Приклад ІФ...

Наступні ТРИ if дають один і той же результат

ВИКОРИСТАННЯ ПОДВІЙНИХ КВАДРАТНИХ ДУЖОК

```
read -p "Do you want to continue?" reply
```

```
if [[ $reply = "y" ]]; then
```

```
    echo "Ви ввели " $reply
```

```
fi
```

ВИКОРИСТАННЯ ОДИНАРНИХ КВАДРАТНИХ ДУЖОК

```
read -p "Do you want to continue?" reply
```

```
if [ $reply = "y" ]; then
```

```
    echo "Ви ввели " $reply
```

```
Fi
```

ВИКОРИСТАННЯ КОМАНДИ TEST

```
read -p "Do you want to continue?" reply
```

```
if test $reply = "y"; then
```

```
    echo "Ви ввели " $reply
```

```
fi
```



Приклад IF...ELIF...

```
#!/bin/bash
```

```
read -p "Введіть дохід: " Income
read -p "Введіть витрати: " Expense
```

```
let Net=$Income-$Expense
if [ "$Net" -eq "0" ]; then
    echo "Дохід і витрати рівні - беззбитковість."
elif [ "$Net" -gt "0" ]; then
    echo "Дохід: " $Net
else
    echo "Витрати: " $Net
fi
```

```
argussss@argussss-VirtualBox:~/BASH$ nano script2.sh
argussss@argussss-VirtualBox:~/BASH$ ./script2.sh
Введіть дохід: 200
Введіть витрати: 100
Дохід: 100
argussss@argussss-VirtualBox:~/BASH$ ./script2.sh
Введіть дохід: 100
Введіть витрати: 100
Дохід і витрати рівні - беззбитковість.
argussss@argussss-VirtualBox:~/BASH$ ./script2.sh
Введіть дохід: 100
Введіть витрати: 200
Витрати: -100
```




Твердження CASE

Використовується для рішень, що мають множинний вибір

Синтаксис:

```
case word in
    pattern1) command-list1
    ;;
    pattern2) command-list2
    ;;
    patternN) command-listN
    ;;
esac
```



Шаблони для CASE

Перевірка на співпадіння для `word`

☐ можуть також мати наступні символи:

`*`

`?`

`[ ... ]` `[:class:]`

☐ Декілька шаблонів перебираються через:

`|`



Приклад для CASE₁

```
#!/bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter q to quit"

read -p "Enter your choice: " reply

case $reply in
    Y|YES) echo "Displaying all (really...) files"
           ls -a ;;
    N|NO)  echo "Display all non-hidden files..."
           ls ;;
    Q)     exit 0 ;;

    *)     echo "Invalid choice!"; exit 1 ;;
esac
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script1.sh
Enter Y to see all files including hidden files
Enter N to see all non-hidden files
Enter q to quit
Enter your choice: N
Displaying all non-hidden files...
script1.sh script.sh
argussss@argussss-VirtualBox:~/BASH$ ./script1.sh
Enter Y to see all files including hidden files
Enter N to see all non-hidden files
Enter q to quit
Enter your choice: q
Invalid choice!
argussss@argussss-VirtualBox:~/BASH$ ./script1.sh
Enter Y to see all files including hidden files
Enter N to see all non-hidden files
Enter q to quit
Enter your choice: Q
```



Приклад для CASE₂

```
#!/bin/bash
ChildRate=3
AdultRate=10
SeniorRate=7
read -p "Enter your age: " age
case $age in
    [1-9] | [1][0-2])    # child, if age 12 and younger
        echo "your rate is" '$' "$ChildRate.00" ;;
    # adult, if age is between 13 and 59 inclusive
    [1][3-9] | [2-5][0-9])
        echo "your rate is" '$' "$AdultRate.00" ;;
    [6-9][0-9])          # senior, if age is 60+
        echo "your rate is" '$' "$SeniorRate.00" ;;
esac
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script.sh
Enter your age: 12
your rate is $3.00
argussss@argussss-VirtualBox:~/BASH$ ./script.sh
Enter your age: 45
your rate is $10.00
argussss@argussss-VirtualBox:~/BASH$ ./script.sh
Enter your age: 68
your rate is $7.00
argussss@argussss-VirtualBox:~/BASH$ ./script.sh 68
Enter your age: 68
your rate is $7.00
argussss@argussss-VirtualBox:~/BASH$
```



Цикл WHILE

Призначення:

Виконання команд з “command-list” поки “expression” є true

Синтаксис:

```
while [ expression ] do  
    command-list done
```



Приклади використання WHILE₁

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
    echo The counter is $COUNTER
    let COUNTER=$COUNTER+1
done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script3.sh
The counter is 0
The counter is 1
The counter is 2
The counter is 3
The counter is 4
The counter is 5
The counter is 6
The counter is 7
The counter is 8
The counter is 9
```



Приклади використання WHILE₂

```
#!/bin/bash
Cont="Y"
while [ $Cont = "Y" ]; do
    ps -A
    read -p "want to continue? (Y/N)" reply
    Cont=`echo $reply | tr [:lower:] [:upper:]`
done
echo "done"
```

```
3128 pts/2      00:00:00 script4.sh
3133 pts/2      00:00:00 ps
want to continue? (Y/N)y
  PID TTY          TIME CMD
    1 ?           00:00:00 systemd
    2 ?           00:00:00 kthreadd
    3 ?           00:00:00 rcu_gp
    4 ?           00:00:00 rcu_par_gp
```



Приклади використання WHILE₃

```
#!/bin/bash
# копіює файли з home- до папки webserver
# нова папка створюється кожну годину
PICSDIR=/home/carol/pics WEBDIR=/var/www/carol/webcam
while true;
do
    DATE=`date +%Y%m%d`
    HOUR=`date +%H`
    mkdir $WEBDIR/"$DATE"
    while [ $HOUR -ne "00" ];
    do
        DESTDIR=$WEBDIR/"$DATE"/"$HOUR"
        mkdir "$DESTDIR"
        mv $PICSDIR/*.jpg "$DESTDIR"/
        sleep 3600
        HOUR=`date +%H`
    done
done
```

```
argussss@argussss-VirtualBox:~/BASH$ tree ~/pics/
/home/argussss/pics/
├── 1.jpg
├── 2.jpg
└── 3.jpg

0 directories, 3 files
```

```
/var/www/argussss/webcam/
├── 20231018
│   └── 10
│       ├── 1.jpg
│       ├── 2.jpg
│       └── 3.jpg
└── 20231019
    ├── 10
    ├── 11
    └── 12

2 directories, 3 files
```




Цикл UNTIL

Призначення:

Виконувати “command-list” доки “expression” є false

Синтаксис:

```
until [ expression ]  
do  
    command-list  
done
```



Приклади використання UNTIL₁

```
#!/bin/bash
```

```
COUNTER=20
until [ $COUNTER -lt 10 ]
do

    echo $COUNTER

    let COUNTER-=1
done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script6.sh
20
19
18
17
16
15
14
13
12
11
10
```



Приклади використання UNTIL₂

```
#!/bin/bash
```

```
Stop="N"
until [ $Stop = "Y" ];
do
    ps -A
    read -p "want to stop? (Y/N)" reply
    Stop=`echo $reply | tr [:lower:] [:upper:]`
done
echo "done"
```

```
3368 ?          00:00:00 kworker/0:1-evil
3369 ?          00:00:00 kworker/0:2
3374 pts/2      00:00:00 script7.sh
3375 pts/2      00:00:00 ps
want to stop? (Y/N) y
done
```



Цикл FOR

Призначення:

Виконання `commands` стільки разів, скільки слів в `"argument-list"`

Синтаксис:

```
for variable in argument-list  
do  
    commands  
done
```



Приклади використання FOR₁

```
#!/bin/bash
```

```
for i in 7 9 2 3 4 5
do
    echo $i
done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script8.sh
7
9
2
3
4
5
```



Приклади використання FOR₂

```
#!/bin/bash
# визначити середню температуру за тиждень

for num in 1 2 3 4 5 6 7
do
    read -p "Введіть temp для дня $num: " Temp
    let TempTotal=TempTotal+Temp
done
let AvgTemp=TempTotal/7
echo "Середня температура: " $AvgTemp
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script9.sh
Введіть temp для дня 1: 12
Введіть temp для дня 2: 16
Введіть temp для дня 3: 20
Введіть temp для дня 4: 27
Введіть temp для дня 5: 27
Введіть temp для дня 6: 19
Введіть temp для дня 7: 17
Середня температура: 19
```



Приклади використання FOR₃

Перебір аргументів командного рядка:

```
#!/bin/bash
for parm
do
    echo $parm
done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script10.sh
argussss@argussss-VirtualBox:~/BASH$ ./script10.sh param1 param2 param3
param1
param2
param3
```



Команда SELECT

Будує просте меню з списку слів

- Дозволяє вводити цифри замість слів
- Користувач вводить порядковий номер замість слова
- Цикл до кінця вводу, тобто ^d (або ^c)

Синтаксис:

```
select WORD in LIST do  
    RESPECTIVE-COMMANDS  
done
```




Приклад використання SELECT₁

```
#!/bin/bash
select var in alpha beta gamma
do
    echo $var
done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script11.sh
1) alpha
2) beta
3) gamma
#? 2
beta
#? 1
alpha
#? 3
gamma
#? y

#? 1
alpha
#? █
```



Приклад використання `SELECT`₂

`PS3` – рядок підказки для `SELECT`

`$REPLY` – вибір користувача

```
#!/bin/bash
```

```
PS3="select entry or ^D: "
```

```
select var in alpha beta
```

```
do
```

```
    echo "$REPLY = $var"
```

```
done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script12.sh
1) alpha
2) beta
select entry or ^D: 1
1 = alpha
select entry or ^D: 2
2 = beta
select entry or ^D: 3
3 =
select entry or ^D: 1
1 = alpha
select entry or ^D: 2
2 = beta
select entry or ^D:
```



Приклад використання SELECT₃

```
#!/bin/bash
echo "Скрипт захисту файлів"
echo "Виберіть файл для захисту:"

select FILENAME in *
do
    echo "Ви обрали $FILENAME ($REPLY) "
    chmod go-rwx "$FILENAME"

    echo "Файл тепер приватний!!!"

done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script13.sh script10.sh
Скрипт захисту файлів
Виберіть файл для захисту:
1) script10.sh   3) script12.sh   5) script1.sh    7) script3.sh   9) script5.sh
2) script11.sh  4) script13.sh   6) script2.sh   8) script4.sh  10) script6.sh
#? 10
Ви обрали script6.sh (10)
Файл тепер приватний!!!
#?
```

601401	12	-rwxr-xr-x	1	argussss	argussss	111	жов	18	10:09	script3.sh
601402	12	-rwxr-xr-x	1	argussss	argussss	158	жов	18	10:16	script4.sh
601403	12	-rwxr-xr-x	1	argussss	argussss	401	жов	18	10:30	script5.sh
601404	12	-rwx-----	1	argussss	argussss	90	жов	18	10:39	script6.sh
601405	12	-rwxr-xr-x	1	argussss	argussss	154	жов	18	10:42	script7.sh



Твердження BREAK та CONTINUE

❑ Переривають цикли `for`, `while` та `until`

❑ Твердження `break`

- передає управління до твердження ПІСЛЯ твердження `done`
- перериває виконання циклу

❑ Твердження `continue`

- передає управління твердженню `done`
- пропускає умову для поточної ітерації
- продовжує виконання циклу



Приклад використання continue та break

```
#!/bin/bash
for index in 1 2 3 4 5 6 7 8 9 10
do
    if [ $index -le 3 ]; then
        echo "continue"
        continue
    fi

    echo $index
    if [ $index -ge 8 ]; then
        echo "break"
        break
    fi
done
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script14.sh
continue
continue
continue
4
5
6
7
8
break
```



Функції в BASH₁

- ❑ Функція оболонки дуже схожа на скрипт
 - зберігає набір команд для подальшого використання
 - оболонка зберігає функції в пам'яті
 - оболонка виконує функцію в тій же самій оболонці, яка її викликала
- ❑ Функцію можна визначити:
 - в `.profile`
 - в скрипті
 - або в командному рядку



Функції в BASH₂

- ☐ Видалення функції
 - за допомогою команди `unset`
- ☐ Повинна бути визначена до її використання
- ☐ Зазвичай визначається на початку скрипта

Синтаксис:

```
function-name () {  
    statements  
}
```



Приклад функції в BASH

```
#!/bin/bash
fun () {
    JUST_A_SECOND=1
    let i=0 REPEATS=30
    echo "And now the fun really begins."
    while [ $i -lt $REPEATS ]
    do
        echo "-----FUNCTIONS are fun----->"
        sleep $JUST_A_SECOND
        let i+=1
    done
}
fun
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script15.sh
And now the fun really begins.
-----FUNCTIONS are fun----->
-----FUNCTIONS are fun----->
-----FUNCTIONS are fun----->
-----FUNCTIONS are fun----->
-----FUNCTIONS are fun----->
-----FUNCTIONS are fun----->
```




Параметри функцій

- ☐ Параметри функції декларувати не потрібно
- ☐ Аргументи функції доступні в тілі функції як \$1, \$2, \$3, ...

\$# відображає кількість параметрів

\$0 відображає назву скрипту (не функції!!!)



Приклади функцій з параметрами₁

```
#!/bin/bash
testfile() {
    if [ $# -gt 0 ]; then
        if [[ -f $1 && -r $1 ]]; then
            echo $1 is a readable file
        else
            echo $1 is not a readable file
        fi
    fi
}
testfile .
testfile funtest
```

```
argussss@argussss-VirtualBox:~/BASH$ sudo ./script16.sh
. is not a readable file
/home/argussss/BASH/funtest is a readable file
```



Приклади функцій з параметрами₂

```
#!/bin/bash
checkfile() {
    for file
    do
        if [ -f "$file" ]; then
            echo "$file is a file"
        else
            if [ -d "$file" ]; then
                echo "$file is a directory"
            fi
        fi
    done
}
checkfile .
checkfile funtest
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script17.sh
. is a directory
funtest is a file
```



Локальні та глобальні змінні функцій

- ☐ Змінні, визначені в функціях, є глобальними
- ☐ Для визначення змінної локальною для функції використовується твердження `local`



Приклад локальних змінних функції

```
#!/bin/bash
```

```
global="pretty good variable"
```

```
foo () {  
    local inside="not so good variable"  
    echo $global  
    echo $inside  
    global="better variable"
```

```
}  
echo $global  
foo  
echo $global  
echo $inside
```

```
argussss@argussss-VirtualBox:~/BASH$ ./script18.sh  
pretty good variable  
pretty good variable  
not so good variable  
better variable
```



Обробка сигналів₁

❑ Linux дозволяє посилати сигнали будь-якому процесу

- -1 = hangup (зависання)
- -2 = перервати через ^C
- без аргументів = перервати
- -9 = вбити

➤ -9 не може бути заблокований

❑ `kill -l`

```
kill -HUP 1234
```

```
kill -2 1235
```

```
kill 1235
```

```
kill -9 1236
```

```
argussss@argussss-VirtualBox:~/BASH$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```



Обробка сигналів₂

- ❑ Дія за замовчуванням для більшості сигналів – припинити процес
- ❑ Bash дозволяє перевизначити як буде оброблятися сигнал
- ❑ Команда `trap` без параметрів відновлює обробку за замовчуванням

Синтаксис:

```
trap 'handler commands' signals
```

Приклад:

```
trap 'echo do not hangup' 1 2
```



Приклади використання `trap`

```
#!/bin/bash
# kill -1 won't kill this process
# kill -2 will

trap 'echo dont hang up' 1

while true
do
    echo "try to hang up"
    sleep 1
done
```




Приклади використання `trap`

```
#!/bin/sh
# plain kill or kill -9 will kill this
trap 'echo 1' 1
trap 'echo 2' 2

while true; do
    echo -n .
    sleep 1
done
```



Приклади використання `trap`

```
#!/bin/bash
trap 'cleanup; exit' 2

cleanup () {
    /bin/rm -f /tmp/tempfile.$$.*
}

for i in 1 2 3 4 5 6 7 8
do
    echo "$i.iteration"
    touch /tmp/tempfile.$$.$i
    sleep 1
done

cleanup
```



Відлагоджування скриптів в BASH

❑ Для відлагоджування використовуються наступні команди:

- `echo`
- `set`

❑ Команда "`set`" має опції для контролю за потоком виконання:

- `-v` друкує кожний рядок, як він прочитаний
- `-x` відображає команду та її аргументи
- `-n` перевіряє на синтаксичні посилки

❑ Опції можна включати або відключати

- Включення: `set -xv`
- Відключення: `set +xv`

❑ Включення через shebang

- `#!/bin/bash -xv`

Донецький національний технічний університет



Тема 3. Bash та Python для DevOps.

Лекція 5. Автоматизація з Python

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

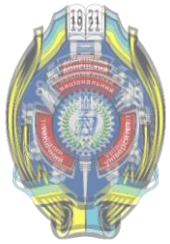
193

- ☐ Мета лекції
- ☐ Основи автоматизації з Python



Мета лекції

- Розглянути принципи роботи з контейнерами



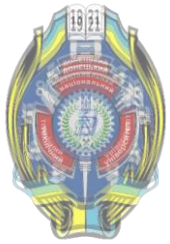
Вступ

- ❑ DevOps - це практика, яка об'єднує розробку програмного забезпечення, операційне обслуговування та безпеку
- ❑ Python - це всебічна мову програмування, яка використовується в різних сферах, включаючи DevOps



Огляд DevOps

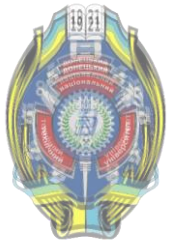
- ❑ DevOps - це методологія, яка об'єднує розробку програмного забезпечення, операційне обслуговування та безпеку
- ❑ Метою DevOps є створення більш ефективного і гнучкого процесу розробки та розгортання програмного забезпечення
- ❑ DevOps фокусується на автоматизації завдань, інтеграції команд і співпраці



Переваги DevOps

□ DevOps може допомогти:

- збільшити швидкість випуску програмного забезпечення
- покращити якість програмного забезпечення
- знизити витрати на обслуговування
- збільшити безпеку програмного забезпечення



Python в DevOps

- ❑ Python - це всебічна мова програмування, яка використовується в різних сферах, включаючи DevOps.
- ❑ Python має ряд переваг для DevOps, включаючи:
 - легкість вивчення і використання
 - широкий вибір бібліотек і інструментів
 - міцна інфраструктура спільноти



Автоматизація завдань

- ❑ Одна з ключових сфер, де Python може використовуватися в DevOps, це автоматизація завдань
- ❑ Python можна використовувати для написання скриптів, які можуть автоматизувати завдання, такі як:
 - створення інфраструктури
 - розгортання програмного забезпечення
 - тестування програмного забезпечення
 - моніторингу і управління



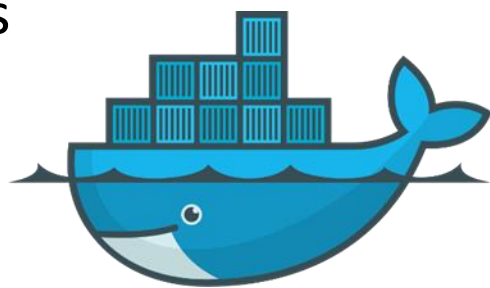
Створення інструментів

- ☐ Python також можна використовувати для створення інструментів, які можуть допомогти в DevOps
- ☐ Ці інструменти можуть бути використані для:
 - автоматизації завдань
 - збору даних
 - аналізу даних
 - розв'язання проблем

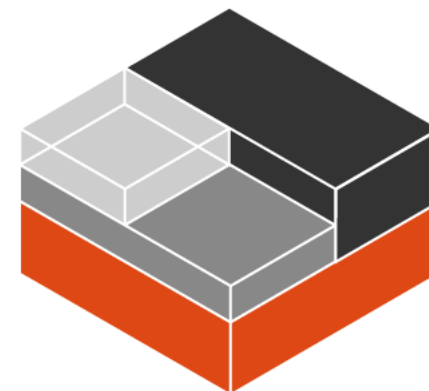


Розширення можливостей платформ DevOps

- ☐ Python також можна використовувати для розширення можливостей платформ DevOps.
- ☐ Це можна зробити, написавши плагіни або розширення для платформ, таких як:
 - Jenkins
 - Ansible
 - Docker
 - Kubernetes



docker





Приклади використання Python в DevOps

- ❑ Ось кілька прикладів того, як Python можна використовувати в DevOps:
 - **Jenkins:** автоматизація запуску тестових наборів або розгортання програмного забезпечення.
 - **Ansible:** автоматизація завдань налаштування серверів або розгортання контейнерів.
 - **Docker:** автоматизація завдань з створення контейнерів або розгортання контейнерів.
 - **Kubernetes:** автоматизація завдань зі створення кластерів Kubernetes або розгортання програмного забезпечення в кластерах Kubernetes.



Практичні приклади використання Python в DevOps

- ☐ Компанія Netflix використовує Python для автоматизації розгортання програмного забезпечення в хмарі.
- ☐ Компанія Spotify використовує Python для моніторингу і управління своїми серверами і мережами.
- ☐ Компанія Google використовує Python для створення своїх інструментів DevOps, таких як Cloud Build і Cloud Deployment Manager.



Автоматизація створення інфраструктури

- ❑ Python можна використовувати для автоматизації створення інфраструктури, наприклад, серверів, мереж, сховищ та інших ресурсів.
- ❑ Це може допомогти організаціям швидко і легко створювати інфраструктуру, необхідну для розробки та розгортання програмного забезпечення.

```
# Створити сервер

import boto3

ec2 = boto3.client('ec2')

response = ec2.create_instances(
    ImageId='ami-
0123456789abcdef0',
    MinCount=1,
    MaxCount=1
)

instance_id =
response['Instances'][0]['InstanceI
d']

print(instance_id)
```




Автоматизація розгортання програмного забезпечення

- ❑ Python можна використовувати для автоматизації розгортання програмного забезпечення, наприклад, на сервери, контейнери або Kubernetes.
- ❑ Це може допомогти організаціям швидко і легко розгорнути програмне забезпечення в виробниче середовище.

```
# Розгорнути програмне забезпечення на сервер
```

```
import paramiko
```

```
ssh = paramiko.SSHClient()
```

```
ssh.connect(hostname='example.com',  
            username='root', password='password')
```

```
sftp = ssh.open_sftp()
```

```
sftp.put('app.tar.gz', '/opt/app.tar.gz')
```

```
ssh.exec_command('tar -xf /opt/app.tar.gz')
```

```
ssh.exec_command('cd /opt/app && ./install.sh')
```

```
ssh.close()
```



Автоматизація тестування програмного забезпечення

- ❑ Python можна використовувати для автоматизації тестування програмного забезпечення, наприклад, функціонального, інтеграційного та системного тестування.
- ❑ Це може допомогти організаціям підвищити якість програмного забезпечення, зменшивши кількість помилок.

```
# Запустити тестовий набір

import unittest

class
MyTestSuite(unittest.TestCase):

    def test_add(self):
        self.assertEqual(1 + 1, 2)

    def test_subtract(self):
        self.assertEqual(1 - 1, 0)

if __name__ == '__main__':
    unittest.main()
```



Автоматизація моніторингу і управління

- ❑ Python можна використовувати для автоматизації моніторингу і управління, наприклад, для моніторингу стану серверів, мереж та іншого обладнання.
- ❑ Це може допомогти організаціям запобігти проблемам і швидко їх вирішувати.

```
# Моніторити стан сервера

import requests

url = 'http://example.com/healthcheck'

response = requests.get(url)

if response.status_code == 200:
    print('Сервер працює нормально')
else:
    print('Сервер не працює')
```



Автоматизація безпеки

- ❑ Python можна використовувати для автоматизації безпеки, наприклад, для сканування мережі на наявність вразливостей.
- ❑ Це може допомогти організаціям захистити свої системи від атак.

Сканувати мережу на наявність вразливостей

```
import nmap
```

```
scanner = nmap.PortScanner()
```

```
scanner.scan('192.168.1.0/24', arguments='-sV -O')
```

```
for host in scanner.all_hosts():  
    for port in scanner[host]['tcp']:  
        if port['state'] == 'open':  
            print(f'Вразливість на  
{host}:{port["portid"]}:  
{port["service"]["version"]}')
```



Автоматизація завдань Jenkins

- ☐ Python можна використовувати для автоматизації завдань Jenkins, наприклад, запуск тестових наборів або розгортання програмного забезпечення.
- ☐ Це може допомогти організаціям покращити ефективність і надійність своїх конвеєрів Jenkins.



Автоматизація завдань Ansible

- ❑ Python можна використовувати для автоматизації завдань Ansible, наприклад, налаштування серверів або розгортання контейнерів.
- ❑ Це може допомогти організаціям уніфікувати свої процеси і покращити продуктивність.

```
# Налаштувати сервер за допомогою
Ansible

import ansible

playbook = 'playbook.yml'

inventory = 'inventory.txt'

results =
ansible.run_playbook(playbook,
inventory)

for host in results['hosts']:
    for task in
results['hosts'][host]['tasks']:
        if task['status'] == 'ok':
            print(f'Налаштування {host}
пройшло успішно')
        else:
            print(f'Налаштування {host}
не пройшло успішно')
```



Автоматизація завдань Docker

- ❑ Python можна використовувати для автоматизації завдань Docker, наприклад, створення контейнерів або розгортання контейнерів.
- ❑ Це може допомогти організаціям покращити керованість і масштабованість своїх контейнерних середовищ.

```
# Створити контейнер Docker
```

```
import docker
```

```
client = docker.from_env()
```

```
image = client.images.build(path='.',  
tag='my-app')
```

```
container =  
client.containers.create(image='my-app')
```

```
container.start()
```



Автоматизація завдань Kubernetes

- ❑ Python можна використовувати для автоматизації завдань Kubernetes, наприклад, створення кластерів Kubernetes або розгортання програмного забезпечення в кластерах Kubernetes.
- ❑ Це може допомогти організаціям покращити керованість і масштабованість своїх Kubernetes-середовищ.

```
# Створити кластер Kubernetes

import kubernetes

client =
kubernetes.client.CoreV1Api()

cluster =
client.create_cluster(
    name='my-cluster',
    kubernetes_version='1.23',
    node_count=3
)
```




Бібліотеки Python для автоматизації завдань

- ❑ Існує безліч бібліотек Python, які можна використовувати для автоматизації завдань.
- ❑ Ось кілька популярних бібліотек:
 - Paramiko: бібліотека для взаємодії з SSH-серверами.
 - Selenium: бібліотека для автоматизації веб-браузерів.
 - Requests: бібліотека для HTTP-запитів.
 - Pytest: бібліотека для тестування програмного забезпечення.
 - Ansible: бібліотека для автоматизації розгортання.
 - Docker: бібліотека для роботи з Docker.
 - Kubernetes: бібліотека для роботи з Kubernetes.



Paramiko

Бібліотека Paramiko
дозволяє взаємодіяти з
SSH-серверами.

Функції:

- Підключення до SSH-сервера.
- Відправка і отримання команд.
- Завантаження і завантаження файлів.

```
import paramiko
```

```
ssh = paramiko.SSHClient()  
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())  
ssh.connect('example.com',  
username='root', password='password')
```

```
# Відправити команду  
ssh.exec_command('ls -la')
```

```
# Отримати результат команди  
output = ssh.recv(1024)
```

```
# Завантажити файл  
sftp = ssh.open_sftp()  
sftp.put('local_file.txt',  
remote_file.txt')
```

```
# Завантажити файл  
sftp.get('remote_file.txt',  
local_file.txt')  
ssh.close()
```



Selenium

❑ Бібліотека Selenium дозволяє автоматизувати веб-браузери.

❑ Функції:

- Відкриття веб-сторінок.
- Введення тексту.
- Натискання кнопок.
- Перевірка стану елементів.

```
from selenium import webdriver
```

```
browser = webdriver.Chrome()  
browser.get('https://google.com')
```

```
# Введіть текст у поле пошуку  
browser.find_element_by_id('q').send_keys('Python')
```

```
# Натисніть кнопку пошуку  
browser.find_element_by_id('btnK').click()
```

```
# Перевірте текст заголовка сторінки  
assert browser.title == 'Python - Пошук Google'
```

```
browser.close()
```



Requests

❑ Бібліотека Requests
дозволяє виконувати HTTP-
запити.

❑ Функції:

- Виконання GET-запитів.
- Виконання POST-запитів.
- Виконання PUT-запитів.
- Виконання DELETE-запитів.

```
import requests
```

```
response =  
requests.get('https://api.github.com/repos/  
bard/python-devops')
```

```
# Перевірте статус коду  
assert response.status_code == 200
```

```
# Отримайте дані з відповіді  
data = response.json()
```

```
# Виведіть ім'я репозиторію  
print(data['name'])
```



Pytest

- ❑ Бібліотека Pytest дозволяє виконувати тестування програмного забезпечення.
- ❑ Функції:
 - Створення тестових наборів.
 - Створення тестових випадків.
 - Визначення очікуваних результатів.



Ansible

- ❑ Бібліотека Ansible дозволяє автоматизувати розгортання.
- ❑ Функції:
 - Налаштування серверів.
 - Розгортання програмного забезпечення.
 - Розгортання контейнерів.

```
# Створити playbook
with open('playbook.yml', 'w') as f:
    f.write('---\n- hosts: all\n  tasks:\n- name: Install Python\n  apt:\n    name=python3 state=present')
```

```
# Запустити playbook
ansible-playbook playbook.yml
```



Docker

❑ Бібліотека Docker дозволяє працювати з Docker.

❑ Функції:

- Створення контейнерів.
- Розгортання контейнерів.
- Керування контейнерами.

```
# Створити контейнер
client = docker.from_env()
container =
client.containers.create(image='python:3.10
')
```

```
# Розгорнути контейнер
container.start()
```

```
# Керувати контейнером
container.stop()
```



Kubernetes

- ❑ Бібліотека Kubernetes дозволяє працювати з Kubernetes.
- ❑ Функції:
 - Створення кластерів Kubernetes.
 - Розгортання програмного забезпечення в кластерах Kubernetes.
 - Керування кластерами Kubernetes.

```
# Створити кластер Kubernetes
client = kubernetes.client.CoreV1Api()
cluster = client.create_cluster(
    name='my-cluster',
    kubernetes_version='1.23',
    node_count=3
)
```

```
# Розгорнути програмне забезпечення в
кластері Kubernetes
client = kubernetes.client.AppsV1Api()
deployment = client.create_deployment(
    name='my-deployment',
    spec=deployment_spec
)
```




Ще кілька бібліотек

- ☐ Jinja2: бібліотека для генерації шаблонів.
- ☐ Faker: бібліотека для генерації штучних даних.
- ☐ Matplotlib: бібліотека для візуалізації даних.
- ☐ NumPy: бібліотека для обробки чисельних даних.
- ☐ Pandas: бібліотека для обробки структурованих даних.



Вибір бібліотеки

- ❑ При виборі бібліотеки для автоматизації завдань слід враховувати наступні фактори:
 - Тип завдань, які потрібно автоматизувати.
 - Наявність необхідних знань і навичок.
 - Доступність бібліотеки.



Заключення

- ☐ Бібліотеки Python можуть бути потужним інструментом для автоматизації завдань DevOps.
- ☐ При виборі бібліотеки слід враховувати тип завдань, які потрібно автоматизувати, наявність необхідних знань і навичок, а також доступність бібліотеки.

Донецький національний технічний університет



Тема 4. Інструментарії віртуалізації та розгортання. Лекція 6. Основи Vagrant

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

225

- ☐ Мета лекції
- ☐ Основи Vagrant



Мета лекції

☐ Розглянути принципи роботи з Vagrant

226



Вступ

- ☐ Vagrant – це інструмент для автоматизації створення та керування віртуальними машинами.
- ☐ Він використовує образ віртуальної машини (VM) як шаблон для створення нової VM.
- ☐ Vagrant може використовуватися для створення VM для будь-якої мети, наприклад, для розробки, тестування або виробництва



Переваги використання Vagrant

- ☐ Простий у використанні
- ☐ Портативний
- ☐ Підтримує широкий спектр платформ
- ☐ Можна використовувати для автоматизації завдань



Історія Vagrant

- ❑ Vagrant був створений в 2010 році Майклом Дойлом.
- ❑ Він був розроблений для спрощення процесу створення та керування віртуальними машинами.



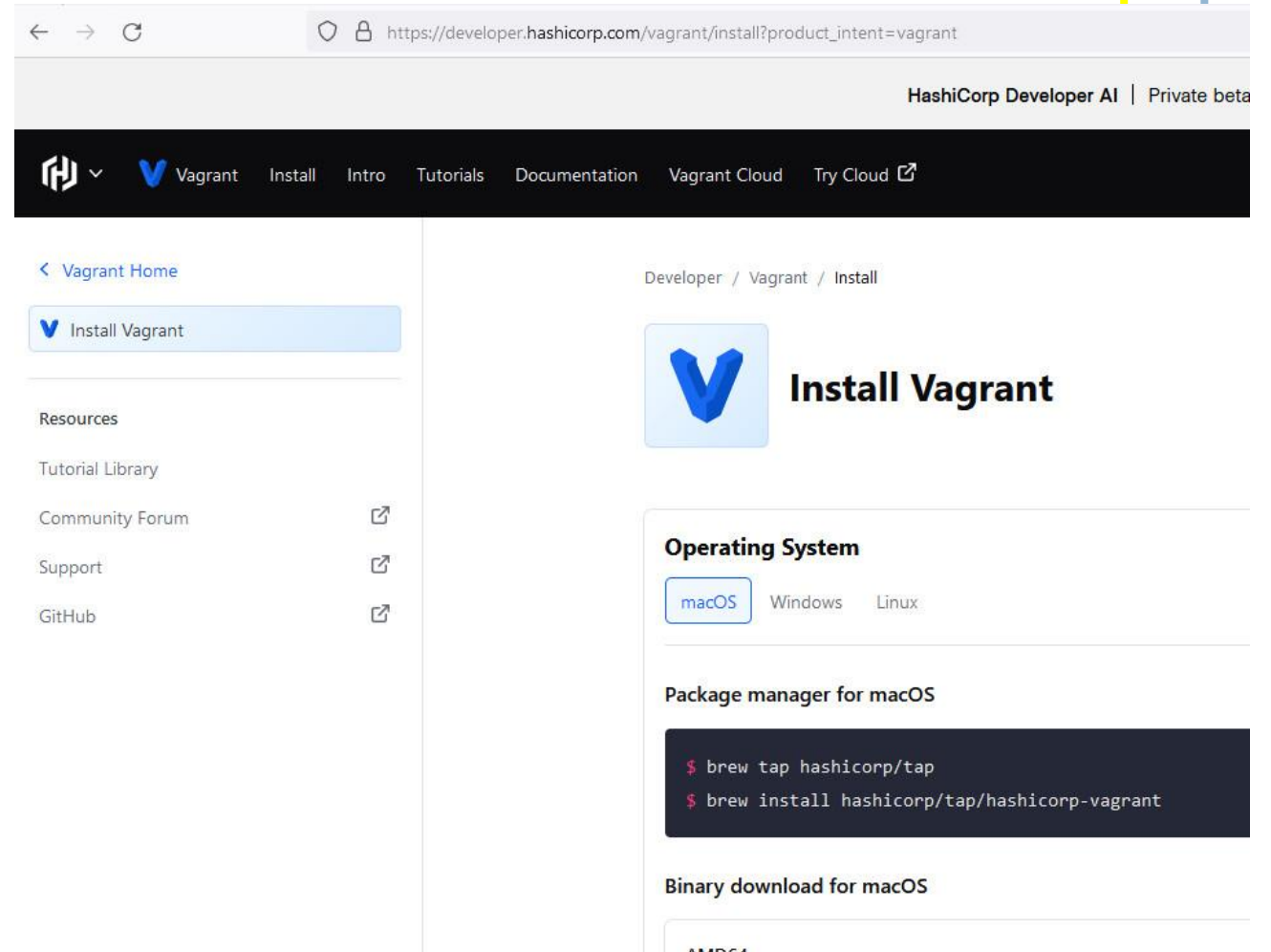
Функціональність Vagrant

- ❑ Vagrant пропонує широкий спектр функцій, які роблять його потужним інструментом для створення та керування віртуальними машинами.
- ❑ До цих функцій належать:
 - Простота використання: Vagrant використовує простий синтаксис для визначення налаштувань VM.
 - Портативність: Vagrant-проекти можна легко переносити з однієї платформи на іншу.
 - Підтримка широкого спектру платформ: Vagrant підтримує різні платформи хостингу, включаючи Windows, macOS та Linux.
 - Автоматизація завдань: Vagrant можна використовувати для автоматизації завдань згортання, налаштування та запуску VM.



Встановлення Vagrant

- ☐ Vagrant доступний для Windows, macOS та Linux.
- ☐ Щоб встановити Vagrant, відвідайте веб-сайт Vagrant і завантажте інсталятор для вашої платформи.





Перший крок: створення проекту

- ❑ Щоб створити проект Vagrant, створіть папку для вашого проекту і створіть у ній файл з розширенням `.vagrantfile`.
- ❑ `mkdir my-project && touch my-project/Vagrantfile` - Створення проекту Vagrant під назвою "my-project".



Вміст файлу Vagrantfile

- ❑ Файл Vagrantfile містить налаштування для вашої VM.
 - ❑ `Vagrant.configure(2) do |config|` - Початок блоку конфігурації.
 - ❑ `config.vm.box = "ubuntu/bionic64"` - Визначення образу VM.
 - ❑ `config.vm.hostname = "my-vm"` - Визначення імені VM.
 - ❑ `end` - Кінець блоку конфігурації.



Визначення box

- ❑ Box - це образ віртуальної машини, який використовується для створення VM.
- ❑ Vagrant має доступ до великої бібліотеки boxів, які можна завантажити з Інтернету.

234

The screenshot shows the Vagrant Cloud website interface. At the top is a blue header with the Vagrant Cloud logo and navigation links for Search and Pricing. Below the header is a section titled "Discover Vagrant Boxes" with a search bar. The search bar contains the text "Search for boxes by operating system, included software, architecture and more". To the right of the search bar is a dropdown menu labeled "Provider". Below the search bar is a list of boxes, each with a logo, name, version, and a list of providers.

Box Name	Version	Providers
ubuntu/trusty64	20190514.0.0	virtualbox
laravel/homestead	13.0.0	hyperv, libvirt, parallels, virtualbox and 2 more providers
hashicorp/precise64	1.1.0	hyperv, virtualbox, vmware_fusion
centos/7	2004.01	hyperv, libvirt, virtualbox, vmware and 3 more providers
ubuntu/xenial64	20211001.0.0	virtualbox



Запуск VM

- ❑ Щоб запустити VM, використовуйте команду:
 - `vagrant up`

```
argus@ArgusM MINGW64 /h/Work/VMs/Vagrant-vm/centosVgVM
$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Checking if box 'eurolinux-vagrant/centos-stream-9' version '9.0.31' is up to date...
==> default: Clearing any previously set forwarded ports...
==> default: Fixed port collision for 22 => 2222. Now on port 2200.
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2200 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2200
    default: SSH username: vagrant
    default: SSH auth method: private key
```



Вхід до VM

- ❑ Щоб увійти до VM, використовуйте команду:
 - `vagrant ssh`

```
argus@ArgusM MINGW64 /h/Work/VMs/Vagrant-vms/centosVgVM
$ vagrant ssh
This box was generated by EuroLinux

Any suggestions are welcome at https://github.com/EuroLinux/cloud-images-rfc/

Happy using.
To delete this message use:
echo '' > /etc/motd
Last login: Tue Oct 31 10:57:09 2023 from 10.0.2.2
[vagrant@centos7 ~]$ |
```




Налаштування VM

- ❑ Можна налаштувати VM, додавши додатковий код до файлу `Vagrantfile`.
- ❑ Наприклад, щоб встановити веб-сервер на VM, додайте наступний код до файлу `Vagrantfile`:

```
Vagrant.configure(2) do
  |config|
    config.vm.box =
      "ubuntu/bionic64"

    config.vm.provision
      "shell", inline: <<-SHELL
        apt-get update
        apt-get install -y
      nginx
      SHELL
    end
```



Доступ до файлів VM

- ❑ Ви можете отримати доступ до файлів VM за допомогою спільного каталогу.
 - `config.vm.synced_folder "./shared",
"/vagrant"` - Створення спільного каталогу між вашим комп'ютером і VM.



Завершення роботи VM

- ❑ Щоб завершити роботу VM, використовуйте:
 - `vagrant halt`

```
argus@ArgusM MINGW64 /h/Work/VMs/Vagrant-vm/centosVgVM  
$ vagrant halt  
==> default: Attempting graceful shutdown of VM...
```



Видалення VM

- ❑ Щоб видалити VM, використовуйте команду:
 - `vagrant destroy`



Додаткові можливості Vagrant

□ Vagrant підтримує безліч додаткових можливостей, таких як:

- *Підтримка різних платформ хостингу*: Vagrant підтримує різні платформи хостингу, включаючи Windows, macOS та Linux.
- *Автоматизація завдань*: Vagrant можна використовувати для автоматизації завдань згортання, налаштування та запуску VM.
- *Розширення*: Vagrant підтримує розширення, які можна використовувати для додавання додаткових функцій.



Приклади використання Vagrant

- ❑ Vagrant можна використовувати для різних цілей, таких як:
 - **Розробка програмного забезпечення:** Vagrant можна використовувати для створення робочого середовища розробки, яке є ізольованим від основного комп'ютера.
 - **Тестування:** Vagrant можна використовувати для створення тестового середовища, яке можна використовувати для тестування програмного забезпечення.
 - **Продакшн:** Vagrant можна використовувати для створення середовища виробництва, яке можна використовувати для розгортання програмного забезпечення.



Базові команди Vagrant

- ❑ Vagrant має набір базових команд, які можна використовувати для управління VM.
- ❑ До цих команд належать:
 - `vagrant up`: Запуск VM.
 - `vagrant halt`: Завершення роботи VM.
 - `vagrant destroy`: Видалення VM.
 - `vagrant ssh`: Вхід до VM.
 - `vagrant reload`: Перезавантаження VM.



Налаштування Vagrantfile

- ❑ Файл Vagrantfile містить налаштування для вашої VM.
- ❑ Можна налаштувати такі параметри у файлі Vagrantfile:

- **Ім'я VM:**

```
config.vm.hostname = "my-vm"
```

- **Розмір VM:**

```
config.vm.provider "virtualbox" do |vb|  
  vb.cpus = 2  
  vb.memory = 2048  
end
```

- **Операційна система VM:**

```
config.vm.box = "ubuntu/bionic64"
```

- **Адреса IP VM:**

```
config.vm.network "public_network",  
  ip: "192.168.1.10"
```




Налаштування мережі

- ❑ Можна налаштувати зберігання для своєї VM.
- ❑ Для цього слід використовувати такі параметри:

- **Публікація VM в Інтернет:**

```
config.vm.network "public_network"
```

- **Підключення VM до приватної мережі:**

```
config.vm.network  
"private_network", ip:  
"192.168.10.10"
```

- **Перенаправлення портів:**

```
config.vm.network forwarded_port",  
guest: 80, host: 8080
```



Налаштування зберігання

❑ Можна налаштувати зберігання для своєї VM, використовуючи такі параметри у файлі `Vagrantfile`:

❑ *Спільний каталог між вашим комп'ютером і VM:*

```
config.vm.synced_folder "./shared", "/vagrant"
```

❑ *Додатковий диск для VM:*

```
config.vm.disk "my_disk", size: 20G
```



Налаштування provision

- ❑ Provision – це процес, який виконується при запуску VM.
- ❑ Можна використовувати provision для установки програмного забезпечення, налаштування параметрів або виконання інших завдань.

- ❑ **Встановлення веб-сервера nginx:**

```
config.vm.provision "shell", inline: <<-SHELL apt-get  
update apt-get install -y nginx SHELL
```

- ❑ **Налаштування бази даних MySQL:**

```
`config.vm.provision "shell", inline: <<-SHELL  
    apt-get update  
    apt-get install -y mysql-server
```

- **Налаштування бази даних**

```
mysql -u root -p < /vagrant/database.sql SHELL
```



Розширення Vagrant

❑ Vagrant підтримує розширення, які можна використовувати для додавання додаткових функцій.

❑ ***Розширення для управління мережами:***

```
vagrant plugin install vagrant-libvirt
```

❑ ***Розширення для управління контейнерами:***

```
vagrant plugin install vagrant-docker
```



Безпека Vagrant

- ❑ Vagrant має набір функцій безпеки, які можна використовувати для захисту ваших VM.
 - Створюйте міцні паролі для ваших VM.
 - Використовуйте мережі ізольовані від основного комп'ютера.
 - Переконайтеся, що ваші VM мають актуальні оновлення безпеки.



Взаємодія з VM

❑ Можна взаємодіяти з VM за допомогою командного рядка або графічної оболонки.

- ***Командний рядок:***

- `vagrant ssh` - Вхід до VM.
- `vagrant provision` - Виконання provision.
- `vagrant destroy` - Видалення VM.

- ***Графічна оболонка:***

- Використовуйте інструменти, такі як VirtualBox або VMware.



Автоматизація завдань

❑ Vagrant можна використовувати для автоматизації завдань згортання, налаштування та запуску VM.

❑ Приклад:

❑ Встановлення веб-сервера nginx:

```
config.vm.provision "shell",  
  inline: <<-SHELL apt-get update apt-get install  
-y nginx SHELL
```



Розгортання програмного забезпечення

- ❑ Vagrant можна використовувати для розгортання програмного забезпечення в середовищі VM.
- ❑ Приклад:
 - Розгортання веб-сайту:
 - Створіть VM з операційною системою Linux.
 - Встановіть веб-сервер nginx на VM.
 - Розгорніть веб-сайт на VM.



Управління конфігурацією

- ❑ Vagrant можна використовувати для управління конфігурацією VM.
- ❑ Приклад:
 - Зміна розміру VM:
 - Змініть параметр "cpus" або "memory" у файлі `Vagrantfile`.



Використання з іншими інструментами

- ❑ Vagrant можна використовувати з іншими інструментами, такими як `Ansible`, `Chef` та `Puppet`.
- ❑ Приклад:
 - Використання `Ansible` для налаштування VM:
 - Додайте в файл `Vagrantfile` команду `"vagrant provision"` з параметром `"ansible"`.
 - Створіть файл `Ansible playbook` для налаштування VM.



Використання в команді

- ❑ Vagrant можна використовувати в команді для створення та керування VM.
- ❑ Приклад: створення VM для кожного розробника:
 - Створіть шаблон `Vagrantfile`.
 - Створіть командний файл для створення VM.
 - Запустіть командний файл для кожного розробника.

```
Vagrant.configure("2") do |config|
```

```
# Встановіть дистрибутив Linux  
config.vm.box = "ubuntu/bionic64"
```

```
# Встановіть Python 3.8  
config.vm.provision "shell", inline: <<-SHELL  
  apt-get update  
  apt-get install -y python3.8  
SHELL
```

```
# Встановіть інші пакети або програми  
# ...
```

```
end
```



Використання в освіті

- ❑ Vagrant можна використовувати в освіті для навчання розробників та адміністраторів систем основам віртуалізації.
- ❑ Приклад: вивчення основ віртуалізації:
 - Створіть VM з операційною системою Linux.
 - Встановіть веб-сервер nginx на VM.
 - Розгорніть веб-сайт на VM.

```
Vagrant.configure("2") do |config|
```

```
# Встановіть дистрибутив Linux  
config.vm.box = "ubuntu/bionic64"
```

```
# Встановіть веб-сервер nginx  
config.vm.provision "shell", inline: <<-SHELL  
  apt-get update  
  apt-get install -y nginx  
SHELL
```

```
end
```

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <title>Веб-сайт</title>  
</head>  
<body>  
  <h1>Вітаю на моєму веб-сайті!</h1>  
</body>  
</html>
```

```
# Розгорніть веб-сайт
```

```
echo "Hello, world!" > /var/www/html/index.html
```



Використання в бізнесі

- ❑ Vagrant можна використовувати в бізнесі для створення робочих середовищ розробки, тестування та виробництва.
- ❑ Приклад: створення робочого середовища розробки:
 - Створіть VM з операційною системою Linux.
 - Встановіть всі необхідні інструменти для розробки на VM.



Використання в хмарній інфраструктурі

- ❑ Vagrant можна використовувати в хмарній інфраструктурі для створення та керування віртуальними машинами в хмарі.
- ❑ Приклад: створення VM на платформі AWS:
 - Створіть шаблон `Vagrantfile`.
 - Створіть командний файл для створення VM.
 - Запустіть командний файл для створення VM на платформі AWS.

```
Vagrant.configure("2") do |config|
```

```
# Тип віртуальної машини
config.vm.provider "aws" do |aws|
  aws.instance_type = "t2.micro"
end
```

```
# Розмір віртуальної машини
config.vm.box_size = "100GB"
```

```
# Операційна система
config.vm.box = "hashicorp/bionic64"
```

```
# Інші пакети або програми
config.vm.provision "shell", inline: <<-SHELL
  apt-get update
  apt-get install -y python3
SHELL
```



Використання в Інтернеті речей

- ❑ Vagrant можна використовувати в Інтернеті речей для створення та керування пристроями IoT.
- ❑ Приклад: створення IoT-пристрою:
 - Створіть VM з операційною системою Linux.
 - Встановіть всі необхідні драйвери та програмне забезпечення для IoT-пристрою

```
Vagrant.configure("2") do |config|
```

```
# Тип віртуальної машини
config.vm.provider "virtualbox" do |vb|
  vb.memory = "1024"
  vb.cpus = 1
end
```

```
# Розмір віртуальної машини
config.vm.box_size = "100GB"
```

```
# Операційна система
config.vm.box = "hashicorp/bionic64"
```

```
# Драйвери та програмне забезпечення
config.vm.provision "shell", inline: <<-SHELL
  sudo apt-get update
  sudo apt-get install -y mosquitto
SHELL
```

```
end
```



Використання в блокчейні

- ❑ Vagrant можна використовувати в блокчейні для створення та керування середовищами блокчейну.
- ❑ Приклад: створення середовища блокчейну:
 - Створіть VM з операційною системою Linux.
 - Встановіть всі необхідні інструменти та програмне забезпечення для блокчейну.

```
Vagrant.configure("2") do |config|
```

```
# Тип віртуальної машини
config.vm.provider "virtualbox" do |vb|
  vb.memory = "1024"
  vb.cpus = 1
end
```

```
# Розмір віртуальної машини
config.vm.box_size = "100GB"
```

```
# Операційна система
config.vm.box = "hashicorp/bionic64"
```

```
# Інструменти та програмне забезпечення
config.vm.provision "shell", inline: <<-SHELL
  sudo apt-get update
  sudo apt-get install -y git docker
SHELL
```

```
end
```




Використання в машинному навчанні

- ❑ Vagrant можна використовувати в машинному навчанні для створення та керування середовищами машинного навчання.
- ❑ Приклад: створення середовища машинного навчання:
 - Створіть VM з операційною системою Linux.
 - Встановіть всі необхідні інструменти та програмне забезпечення для машинного навчання.

```
Vagrant.configure("2") do |config|
```

```
# Тип віртуальної машини
config.vm.provider "virtualbox" do |vb|
  vb.memory = "1024"
  vb.cpus = 1
end
```

```
# Розмір віртуальної машини
config.vm.box_size = "100GB"
```

```
# Операційна система
config.vm.box = "hashicorp/bionic64"
```

```
# Інструменти та пакети для машинного навчання
config.vm.provision "shell", inline: <<-SHELL
  sudo apt-get update
  sudo apt-get install -y python3-pip
  pip3 install tensorflow
SHELL
```

```
end
```



Використання в штучному інтелекті

- ❑ Vagrant можна використовувати в штучному інтелекті для створення та керування середовищами штучного інтелекту.
- ❑ Приклад: створення середовища штучного інтелекту:
 - Створіть VM з операційною системою Linux.
 - Встановіть всі необхідні інструменти та програмне забезпечення для штучного інтелекту.

```
Vagrant.configure("2") do |config|  
  
  # Тип віртуальної машини  
  config.vm.provider "virtualbox" do |vb|  
    vb.memory = "1024"  
    vb.cpus = 1  
  end  
  
  # Розмір віртуальної машини  
  config.vm.box_size = "100GB"  
  
  # Операційна система  
  config.vm.box = "hashicorp/bionic64"  
  
  # Інструменти та пакети для ШІ  
  config.vm.provision "shell", inline: <<-SHELL  
    sudo apt-get update  
    sudo apt-get install -y python3-pip  
    pip3 install tensorflow  
    pip3 install keras  
    pip3 install scikit-learn  
  SHELL  
end
```



Приклад реального конфігу

Цей Vagrantfile створює віртуальну машину з Linux, яка використовується для розробки та тестування великого реального проекту.

Тип віртуальної машини

```
config.vm.provider "virtualbox" do |vb|  
  vb.memory = "10240"  
  vb.cpus = 4  
end
```

Розмір віртуальної машини

```
config.vm.box_size = "1000GB"
```

Операційна система

```
config.vm.box = "hashicorp/bionic64"
```

Інструменти та пакети для розробки та тестування

```
config.vm.provision "shell", inline: <<-SHELL  
  sudo apt-get update  
  sudo apt-get install -y git
```

```
sudo apt-get install -y python3-pip  
sudo apt-get install -y docker
```

Встановіть потрібні інструменти та пакети

```
pip3 install -r requirements.txt
```

Налаштуйте середовище

```
echo "MY_ENV_VAR=12345" >> .env  
SHELL
```

Налаштування мережі

```
config.vm.network "private_network", ip: "192.168.1.100"
```

Налаштування портів

```
config.vm.network "forwarded_ports", guest: 80, host: 8080
```

Налаштування хостингу

```
config.vm.hostname = "my-project"
```



Заключення

- ❑ Vagrant – це потужний інструмент, який можна використовувати для створення та керування віртуальними машинами.
- ❑ Vagrant простий у використанні і підтримує широкий спектр платформ і можливостей.

Донецький національний технічний університет



Тема 4. Інструментарії віртуалізації та розгортання. Лекція 7. Основи Capistrano

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

266

- ☐ Мета лекції
- ☐ Основи Capistrano



Мета лекції

- Розглянути принципи роботи з Capistrano

267



Загальний огляд Capistrano

❑ Що таке Capistrano?

- Capistrano – це інструмент розгортання для веб-застосунків.
- Він написаний на Ruby і використовує SSH для підключення до серверів.
- Capistrano можна використовувати для автоматизації завдань розгортання, таких як копіювання файлів, налаштування веб-сервера та перезапуск веб-застосунків.



Застосування і вигоди в розробці

❑ Застосування Capistrano

- Capistrano можна використовувати для розгортання веб-застосунків на будь-якому типі сервера.
- Він часто використовується для розгортання веб-застосунків на хмарних серверах, таких як Amazon Web Services (AWS) або Google Cloud Platform (GCP).

❑ Вигоди Capistrano

- Capistrano може допомогти вам автоматизувати завдання розгортання, що може заощадити час і зменшити ймовірність помилок.
- Capistrano може допомогти вам стандартизувати процеси розгортання, що може покращити надійність і повторюваність ваших розгортань.
- Capistrano може допомогти вам покращити комунікацію між членами команди, що може полегшити спільну роботу над розгортаннями.



Встановлення Capistrano

☐ Вимоги до встановлення

- Ruby версії 2.7 або новішої
- SSH

☐ Команда встановлення

- `gem install capistrano`



Створення нового проекту

- ☐ Щоб створити новий проект Capistrano, використовуйте наступну команду:
`capify` .
- ☐ Команда `capify` використовується для створення нового проекту Capistrano. Команда `capify` створює новий файл `Capfile` у поточному каталозі.



Основні компоненти Capistrano

- ☐ `Capfile` – це головний файл Capistrano, який визначає параметри розгортання.
- ☐ `Roles` – це визначення ролей серверів, які будуть розгортатися.
- ☐ `Tasks` – це завдання, які Capistrano виконує під час розгортання.
- ☐ `Hooks` – це точки в розгортанні, в яких можна вставити власний код.



Конфігурація Capistrano

Структура конфігураційних файлів

❑ Конфігурація Capistrano зберігається в файлах з розширенням `.rb`. Ці файли зберігаються в каталозі `config/` проекту Capistrano.

Основні файли конфігурації Capistrano:

❑ `Capfile` – це головний файл Capistrano, який визначає параметри розгортання.

❑ `Roles` – це файл, який визначає ролі серверів, як будуть розгортатися.

❑ `Tasks` – це файл, який визначає завдання, які Capistrano виконує під час розгортання.

❑ `Hooks` – це файл, який визначає хуки Capistrano.



Змінні та їх призначення

- ❑ Capistrano використовує змінні для зберігання інформації про розгортання. Змінні можна визначати в файлі `Capfile` або в інших файлах конфігурації.
- ❑ Ось деякі з найпоширеніших змінних Capistrano:
 - `app_name` – ім'я проекту.
 - `stage` – середовище розгортання.
 - `servers` – список серверів, які будуть розгортатися.
 - `roles` – список ролей серверів.



Приклад базового конфігураційного файлу

Цей файл визначає наступне:

- ☐ **Ім'я проекту:** my_app
- ☐ **Середовище розгортання:**
production
- ☐ **Список серверів:**
 - my-web-server.example.com – роль web
 - my-app-server.example.com – роль app
 - my-db-server.example.com – роль db

```
require 'capistrano/setup'
```

```
# Налаштовуємо ім'я проекту та середовище  
розгортання
```

```
set :app_name, 'my_app'
```

```
set :stage, 'production'
```

```
# Налаштовуємо список серверів
```

```
role :web, 'my-web-server.example.com'
```

```
role :app, 'my-app-server.example.com'
```

```
role :db, 'my-db-server.example.com'
```

```
# Завантажуємо стандартні завдання Capistrano
```

```
require 'capistrano/deploy'
```



Основні команди Capistrano₁

- ❑ `cap deploy`
 - Основна команда для розгортання. Вона виконує всі завдання, які визначені для розгортання.
- ❑ Команда `cap deploy` виконує наступні завдання:
 - Копіює файли з репозиторія на сервери.
 - Налаштовує веб-сервер.
 - Перезапускає веб-застосунок.



Основні команди Capistrano₂

- ❑ `cap rollback`
 - ❑ Відкат попереднього розгортання. Вона виконує всі завдання, які визначені для відкату.
- ❑ Команда `cap rollback` виконує наступні завдання:
 - Видаляє файли з серверів.
 - Налаштовує веб-сервер на попереднє значення.
 - Перезапускає веб-застосунок на попереднє значення.



Основні команди Capistrano₃

□ Інші корисні команди

- `cap deploy:check` – перевіряє, чи все готове до розгортання.
- `cap deploy:dry-run` – виконує розгортання в режимі налагодження.
- `cap deploy:restart` – перезапускає веб-додаток без копіювання файлів.
- `cap deploy:rollback:check` – перевіряє, чи все готове до відкату.
- `cap deploy:rollback:dry-run` – виконує відкат в режимі налагодження.



З'єднання з Серверами

- ❑ Capistrano використовує SSH для підключення до серверів. SSH – це протокол, який забезпечує безпечне з'єднання між комп'ютерами.
- ❑ Щоб з'єднатися з сервером за допомогою SSH, потрібно мати ключ аутентифікації. Ключ аутентифікації – це файл, який містить публічні та приватні ключі.
- ❑ Ваш публічний ключ потрібно додати до файлу `authorized_keys` на сервері. Це дозволить вам з'єднатися з сервером без необхідності вводити пароль.



Конфігурація серверів у Capistrano

- ❑ Щоб Capistrano міг з'єднатися з серверами, потрібно налаштувати список серверів у файлі `Capfile`.
- ❑ Приклад налаштування серверів у файлі `Capfile`:
 - `role :web, 'my-web-server.example.com'`
 - `role :app, 'my-app-server.example.com'`
 - `role :db, 'my-db-server.example.com'`



Додавання серверів у Capistrano

- ❑ Файл `servers.rb` є файлом налаштування, який містить список серверів, які потрібно розгорнути. Файл `servers.rb` знаходиться в кореневій папці проекту Capistrano.
- ❑ Щоб додати сервер до файлу `servers.rb`, використовуйте метод `server()`. Метод `server()` приймає два аргументи:
 - Ім'я або IP-адреса сервера.
 - Налаштування сервера.
- ❑ Приклад визначення сервера в файлі `servers.rb`:

```
server 'example.com', user: 'deployer',  
roles: %w[app db web]
```



Приклади додавання серверів у Capistrano

❑ **Сервер з іменем і ролями:**

```
server 'example.com', roles: %w[app db web]
```

❑ **Сервер з ім'ям, користувачем і ролями:**

```
server 'example.com', user: 'deployer', roles: %w[app db web]
```

❑ **Сервер з ім'ям, користувачем, ролями і налаштуваннями SSH:**

```
server 'example.com', user: 'deployer', roles: %w[app db web], ssh_options: {  
  forward_agent: true,  
  port: 2222,  
}
```

❑ **Сервер з IP-адресою і ролями:**

```
server '192.168.1.100', roles: %w[app db web]
```

❑ **Сервер з IP-адресою, користувачем і ролями:**

```
server '192.168.1.100', user: 'deployer', roles: %w[app db web]
```



Додавання шаблонів у Capistrano

- ❑ Щоб додати шаблони до Capistrano, використовуйте файл `deploy.rb`.
- ❑ Файл `deploy.rb` є файлом налаштування, який містить список шаблонів, які потрібно використовувати для розгортання веб-додатків. Файл `deploy.rb` знаходиться в кореневій папці вашого проекту Capistrano.
- ❑ Щоб додати шаблон до файлу `deploy.rb`, використовуйте метод `set()`. Метод `set()` приймає два аргументи:
 - Ім'я шаблону.
 - Налаштування шаблону.
- ❑ Приклад визначення шаблону в файлі `deploy.rb`:

```
set :deploy_via, :remote_cache
set :rvm_ruby_version, '2.7.1'
```
- ❑ Цей код визначає шаблон під назвою `default`. Він використовує метод `remote_cache` для розгортання веб-застосунку. Він також встановлює версію Ruby для використання



Приклади додавання шаблонів у Capistrano₁

- ❑ **Шаблон з ім'ям і методом розгортання:**

```
set :deploy_via, :copy
```

- ❑ **Шаблон з ім'ям, методом розгортання і версією Ruby:**

```
set :deploy_via, :remote_cache  
set :rvm_ruby_version, '2.7.1'
```

- ❑ **Шаблон з ім'ям, методом розгортання, директорією веб-додатку, директорією синхронізації і шляхом розгортання:**

```
set :deploy_via, :remote_cache  
set :app_dir, '/var/www/myapp'  
set :shared_dir, '/var/www/shared'  
set :release_path, '/var/www/releases'
```




Приклади додавання шаблонів у Capistrano₂

- ❑ Шаблон з ім'ям, методом розгортання, директорією веб-додатку, директорією синхронізації, шляхом розгортання і списком завдань, які потрібно виконати перед і після розгортання:

```
set :deploy_via, :remote_cache
set :app_dir, '/var/www/myapp'
set :shared_dir, '/var/www/shared'
set :release_path, '/var/www/releases'

before 'deploy', 'check_revision'
after 'deploy:symlink', 'restart_app'
```



Використання різних середовищ (production, staging)

- ☐ Capistrano дозволяє вам використовувати різні середовища для розгортання. Наприклад, можна мати середовище розробки, середовище тестування та середовище виробництва.
- ☐ Щоб використовувати різні середовища, потрібно визначити середовище в файлі `Capfile`.
- ☐ Приклад налаштування середовища в файлі `Capfile`:
 - `set :stage, :production`
- ☐ Цей код визначає середовище як `production`.
- ☐ Можна налаштувати різні завдання для кожного середовища в файлі `Tasks`.



Управління залежностями та бібліотеками

Використання Gemfile та Bundler

- ❑ `Gemfile` – це файл, який містить список бібліотек, які веб-застосунок використовує.
- ❑ `Bundler` – це інструмент, який використовується для завантаження та установки бібліотек, визначених у `Gemfile`.
- ❑ Щоб використовувати `Gemfile` і `Bundler`, потрібно створити файл `Gemfile` і додати до нього список бібліотек, які ваш веб-додаток використовує.

- ❑ Приклад файлу `Gemfile`:

```
source 'https://rubygems.org'
gem 'rails', '~> 6.1.4'
gem 'pg', '~> 1.2.3'
gem 'puma', '~> 5.3.1'
```



Підтримка різних версій Ruby та гемів

- ❑ Bundler дозволяє підтримувати різні версії Ruby та гемів. Це може бути корисно, якщо використовується веб-застосунок, який вимагає певної версії Ruby або гему.
- ❑ Щоб підтримувати різні версії Ruby та гемів, можна використовувати файл `.ruby-version`. Файл `.ruby-version` визначає версію Ruby, яку потрібно використовувати.
- ❑ Приклад файлу `.ruby-version`:
`2.7.3`
- ❑ Цей код визначає версію Ruby як 2.7.3.
- ❑ Можна використовувати файл `Gemfile.lock` для підтримки різних версій гемів. Файл `Gemfile.lock` визначає конкретні версії гемів, які Bundler повинен використовувати.



Оновлення та управління версіями бібліотек

❑ Bundler дозволяє оновлювати бібліотеки, визначені у `Gemfile`. Можна використовувати команду `bundle update` для оновлення всіх бібліотек або команду `bundle update [gem-name]` для оновлення конкретної бібліотеки.

❑ Приклад:

```
bundle update rails
```

❑ `bundle outdated`

❑ `bundle show [gem-name]`



Робота із змінними середовища

Використання змінних оточення

- ☐ Змінні середовища – це змінні, які зберігаються в середовищі виконання. Вони доступні для всіх процесів, які працюють у цьому середовищі.
- ☐ Змінні середовища можна використовувати для зберігання конфігурацій, ключів доступу та інших даних, які потрібно використовувати різними процесами.
- ☐ Синтаксис визначення змінної оточення:
`export [variable-name]=[value]`
- ☐ Приклад:
`export
DATABASE_URL=postgres://my_user:my_password@my_db_server:5432/my_database`
- ☐ Цей код визначає змінну середовища `DATABASE_URL`, яка містить URL-адресу бази даних.



Захист конфіденційної інформації

- ☐ Змінні середовища можуть використовуватися для зберігання конфіденційної інформації, наприклад, ключів доступу до баз даних або API.
- ☐ Щоб захистити конфіденційну інформацію в змінних середовища, можна використовувати файл `.env`. Файл `.env` – це прихований файл, який зберігається в домашньому каталозі користувача.
- ☐ Щоб використовувати файл `.env`, потрібно включити його в свій шлях середовища. Можна зробити це, додавши наступний код до свого файлу `.bashrc` або `.zshrc`:

```
export PATH=$PATH:$HOME/.env
```
- ☐ Цей код додає каталог `.env` до шляху середовища.



Передача параметрів через командний рядок

- ❑ Можна використовувати змінні середовища для передачі параметрів через командний рядок. Це може бути корисно, якщо потрібно розгорнути різні версії веб-додатку на різних серверах.
- ❑ Щоб передати параметр через командний рядок, використовуйте наступний синтаксис:

```
cap [stage] deploy [parameter]=[value]
```

- ❑ Приклад:

```
cap production deploy  
DATABASE_URL=postgres://my_user:my_password@my_db_server:5432/my_database
```

- ❑ Цей код розгортає веб-застосунок у середовищі `production` і використовує змінну середовища `DATABASE_URL` для визначення URL-адреси бази даних.



Робота з базами даних

Підтримка різних систем управління базами даних (DBMS)

- ❑ Capistrano підтримує різні системи управління базами даних (DBMS), такі як PostgreSQL, MySQL, Oracle та SQLite.
- ❑ Щоб використовувати Capistrano для управління базами даних, потрібно визначити тип DBMS у файлі `Capfile`.
- ❑ Приклад визначення типу DBMS у файлі `Capfile`:

```
set :dbms, :postgresql
```
- ❑ Цей код визначає тип DBMS як PostgreSQL.



Використання capistrano-db-tasks для керування базами даних

- ❑ capistrano-db-tasks – це гем, який надає додаткові завдання для роботи з базами даних у Capistrano.
- ❑ Щоб використовувати capistrano-db-tasks, потрібно встановити його за допомогою наступної команди:
`gem install capistrano-db-tasks`
- ❑ Після встановлення capistrano-db-tasks можна використовувати його завдання для управління базами даних.
- ❑ Приклад використання завдання для створення бази даних:
`cap db:create`
- ❑ Цей код створює базу даних із ім'ям `my_database`.



Робота з міграціями та оновленнями

- ❑ Міграції – це інструмент, який використовується для зміни структури бази даних. Capistrano підтримує міграції за допомогою завдання `cap db:migrate`.
- ❑ Щоб використовувати завдання `cap db:migrate`, потрібно створити файл `db/migrate/[timestamp].rb` для кожної міграції. Файл міграції повинен містити код, який виконує зміни в базі даних.
- ❑ Приклад файлу міграції:

```
class CreateUsers < ActiveRecord::Migration[6.1]
  def change
    create_table :users do |t|
      t.string :name
      t.string :email
      t.string :password_digest

      t.timestamps
    end
  end
end
```



Налаштування резервного копіювання

- Файл `config/deploy/backup.rb` містить налаштування для резервного копіювання. Налаштування резервного копіювання включають в себе наступні параметри:
- `keep_releases` (обов'язково): Кількість випусків, які потрібно зберегти.
 - `backup_path` (обов'язково): Директорія, в якій будуть зберігатися резервні копії.
 - `exclude` (опціонально): Список файлів і директорій, які потрібно виключити з резервної копії.
 - `include` (опціонально): Список файлів і директорій, які потрібно включити в резервну копію.



Резервне копіювання на основі розгортання

- ❑ Capistrano також дозволяє робити резервне копіювання на основі розгортання. Щоб зробити резервне копіювання на основі розгортання, виконайте наступні дії:
 1. Налаштуйте файл `config/deploy/backup.rb`. Цей файл містить налаштування для резервного копіювання.
 2. Додайте параметр `on` до завдання резервного копіювання. Параметр `on` визначає, коли потрібно виконувати завдання резервного копіювання.
 3. Запустіть команду `cap backup`. Ця команда виконає резервне копіювання всіх веб-застосунків, які визначені в файлі `config/deploy/production.rb`.



Відновлення

- ❑ Capistrano включає в себе вбудований механізм відновлення. Щоб відновити веб-застосунок за допомогою Capistrano, виконайте наступні дії:
 1. Налаштуйте файл `config/deploy/restore.rb`. Цей файл містить налаштування для відновлення.
 2. Запустіть команду `cap restore`. Ця команда відновить веб-застосунок із резервної копії, яка вказана в налаштуваннях.



Налаштування відновлення

- Файл `config/deploy/restore.rb` містить налаштування для відновлення. Налаштування відновлення включають в себе наступні параметри:
- `release_path` (обов'язково): Директорія, в якій буде розгорнута резервна копія.
 - `backup_path` (обов'язково): Директорія, в якій зберігаються резервні копії.
 - `revision` (опціонально): Номер випуску, який потрібно відновити. Якщо цей параметр не вказаний, буде відновлена найостанніша резервна копія.



Приклад налаштування відновлення

```
set :release_path, '/var/www/releases'  
set :backup_path, '/var/backups/myapp'
```

❑ Цей код відновлює веб-застосунок із резервної копії, яка зберігається в директорії `/var/backups/myapp`.

❑ Щоб відновити веб-застосунок, виконайте наступну команду:

```
cap restore
```

❑ Ця команда відновить веб-застосунок із резервної копії, яка вказана в налаштуваннях.



Відновлення на основі розгортання

Capistrano також дозволяє відновлювати веб-застосунки на основі розгортання. Щоб відновити веб-застосунок на основі розгортання, виконайте наступні дії:

1. Налаштуйте файл `config/deploy/restore.rb`. Цей файл містить налаштування для відновлення.
2. Додайте параметр `on` до завдання відновлення. Параметр `on` визначає, коли потрібно виконувати завдання відновлення.
3. Запустіть команду `cap restore`. Ця команда відновить веб-додаток із резервної копії, яка вказана в налаштуваннях.



Розширення Capistrano

- ☐ Capistrano є дуже гнучким інструментом, який можна легко розширити за допомогою плагінів. Плагіни Capistrano можуть додавати нові функції, такі як підтримка нових серверів або технологій.
- ☐ Плагіни Capistrano можна знайти в репозиторії RubyGems. Щоб знайти плагіни, які відповідають вашим потребам, можна скористатися пошуком RubyGems.
- ☐ **Щоб встановити плагін Capistrano, виконайте наступні дії:**
 1. Відкрийте термінал і перейдіть в директорію, в якій знаходиться проект Capistrano.
 2. Виконайте команду `gem install [назва-плагіна]`.
- ☐ Наприклад, щоб встановити плагін `capistrano-passenger`, виконайте наступну команду:
`gem install capistrano-passenger`
- ☐ Після встановлення плагіна вам потрібно налаштувати його відповідно до ваших потреб. Налаштування плагінів Capistrano зазвичай виконується в файлі `config/deploy/[назва-плагіна].rb`.



Відомі плагіни Capistrano

- ❑ `capistrano-passenger`: плагін для підтримки веб-сервера Passenger.
- ❑ `capistrano-rvm`: плагін для управління версіями Ruby.
- ❑ `capistrano-bundler`: плагін для управління залежностями Ruby.
- ❑ `capistrano-sidekiq`: плагін для управління Sidekiq.



Плагін `capistrano-passenger`

- ❑ Плагін `capistrano-passenger` забезпечує підтримку веб-сервера Passenger. Passenger – це веб-сервер, який дозволяє запускати Ruby on Rails веб-застосунки на Apache або Nginx.
- ❑ Плагін `capistrano-passenger` додає наступні можливості:
 - Автоматичне встановлення Passenger на сервери.
 - Автоматичне створення конфігурації Passenger для веб-застосунків.
 - Автоматичне перезавантаження Passenger після розгортання веб-застосунку.



Плагін `capistrano-rvm`

- ❑ Плагін `capistrano-rvm` забезпечує управління версіями Ruby. RVM – це менеджер версій Ruby, який дозволяє встановлювати і перемикатися між різними версіями Ruby.
- ❑ Плагін `capistrano-rvm` додає наступні можливості
 - Автоматичне встановлення RVM на сервери.
 - Автоматичне встановлення потрібної версії Ruby на сервери.
 - Автоматичне перемикання на потрібну версію Ruby перед розгортанням веб-застосунку.



Плагін `capistrano-bundler`

- ❑ Плагін `capistrano-bundler` забезпечує управління залежностями Ruby. Bundler – це менеджер залежності Ruby, який дозволяє встановлювати і керувати залежностями Ruby для веб-застосунків.
- ❑ Плагін `capistrano-bundler` додає наступні можливості:
 - Автоматичне встановлення Bundler на сервери.
 - Автоматичне встановлення залежностей Ruby для веб-застосунків.
 - Автоматичне оновлення залежностей Ruby для веб-застосунків.



Плагін `capistrano-sidekiq`

- ❑ Плагін `capistrano-sidekiq` забезпечує управління Sidekiq. Sidekiq – це розбиття завдань для Ruby on Rails веб-застосунків.
- ❑ Плагін `capistrano-sidekiq` додає наступні можливості:
 - Автоматичне встановлення Sidekiq на сервери.
 - Автоматичне створення конфігурації Sidekiq для веб-застосунків.
 - Автоматичне перезавантаження Sidekiq після розгортання веб-застосунку.



Плагін Capistrano-rails

❑ Плагін `capistrano-rails` – це плагін Capistrano, який забезпечує підтримку розгортання веб-додатків на Ruby on Rails. Плагін включає в себе шаблони для розгортання веб-застосунків Rails на різних типах серверів, включаючи:

- Apache з Passenger
- Nginx з Passenger
- Unicorn
- Puma

❑ Плагін `capistrano-rails` також включає в себе ряд додаткових функцій, які спрощують розгортання веб-застосунків Rails, таких як:

- Автоматичне встановлення гемів Ruby on Rails
- Автоматичне створення конфігурації веб-сервера для веб-додатків Rails
- Автоматичне перезавантаження веб-сервера після розгортання веб-додатку



Інші популярні плагіни

Ось деякі інші популярні плагіни Capistrano:

- ❑ `capistrano-nginx`: плагін для підтримки веб-сервера Nginx.
- ❑ `capistrano-postgresql`: плагін для підтримки бази даних PostgreSQL.
- ❑ `capistrano-mysql`: плагін для підтримки бази даних MySQL.
- ❑ `capistrano-redis`: плагін для підтримки Redis.
- ❑ `capistrano-whenever`: плагін для управління завданнями cron.
- ❑ `capistrano-scm-git`: плагін для підтримки Git для управління версіями.



Інтеграція з іншими інструментами

З'єднання з Ansible, Chef, Puppet

- ❑ Capistrano можна інтегрувати з іншими інструментами для автоматизації, такими як Ansible, Chef і Puppet. Це дозволяє використовувати сильні сторони кожного інструменту для створення більш ефективного процесу розгортання.
- ❑ Ось кілька способів інтеграції Capistrano з іншими інструментами:
 - Можна використовувати завдання Capistrano для запуску завдань інших інструментів.
 - Можна використовувати змінні середовища для обміну інформацією між інструментами.
 - Можна використовувати API інших інструментів для виконання завдань у Capistrano.



Використання Capistrano зі збірниками контейнерів, такими як Docker

- ❑ Capistrano можна використовувати для розгортання веб-додатків, які розміщені у контейнерах Docker. Це дозволяє створювати масштабовані та переносні розгортання.
- ❑ Ось кілька способів використання Capistrano зі збірниками контейнерів:
 - Можна використовувати завдання Capistrano для створення, запуску та зупинки контейнерів Docker.
 - Можна використовувати змінні середовища для передачі параметрів контейнерам Docker.
 - Можна використовувати API Docker для виконання завдань у Capistrano.



Інші можливі інтеграції

- ❑ Крім Ansible, Chef і Puppet, Capistrano можна інтегрувати з іншими інструментами, такими як:
 - Ansible Tower – це платформа управління Ansible, яка дозволяє автоматизувати завдання на серверах.
 - Chef Habitat – це платформа управління Chef, яка дозволяє створювати та розгортати контейнери.
 - Puppet Enterprise – це платна версія Puppet, яка надає додаткові функції та підтримку.
 - Terraform – це інструмент для управління інфраструктурою, який дозволяє створювати та керувати серверами, мережами та іншими ресурсами.



Використання Capistrano для відстеження помилок

Логування важливих подій

- ❑ Логування – це процес запису важливих подій, які відбуваються в системі. Логування може бути корисним для відстеження помилок, виявлення проблем і аналізу продуктивності.
- ❑ Ось кілька важливих подій, які слід логувати:
 - Помилки – всі помилки, які виникають у системі, повинні бути логовані. Це допоможе визначити причину помилок і знайти рішення.
 - Зміни конфігурації – всі зміни конфігурації системи повинні бути логовані. Це допоможе відстежувати зміни в системі і вирішувати конфлікти.
 - Виконання завдань – всі завдання, які виконуються в системі, повинні бути логовані. Це допоможе відстежувати прогрес і виявляти проблеми.



Забезпечення безпеки в Capistrano

Захист доступу до конфігурацій та коду

❑ Capistrano дозволяє захистити доступ до конфігурацій та коду за допомогою змінних середовища. Можна використовувати змінні середовища для зберігання паролів та інших конфіденційних даних.

❑ Приклад використання змінних середовища для захисту доступу до конфігурацій:

```
# У файлі Capfile
set :database_password, ENV['DATABASE_PASSWORD']
# У завданні deploy:setup
run "sudo -u postgres psql -c \"ALTER USER my_user WITH PASSWORD
'#{database_password}'\""
```

❑ У цьому прикладі змінна середовища DATABASE_PASSWORD використовується для зберігання пароля бази даних. За допомогою завдання `deploy:setup` пароль бази даних задається для користувача `my_user`.



Порівняння з альтернативами

❑ Capistrano – це один з багатьох інструментів, які можна використовувати для автоматизації розгортання веб-застосунків. Ось деякі альтернативи Capistrano:

- Fabric – це інший популярний інструмент для автоматизації розгортання, який схожий на Capistrano за багатьма аспектами. Однак Fabric має деякі ключові відмінності, такі як його акцент на використанні сценаріїв Ruby.
- Jenkins – це платформа автоматизації, яка може використовуватися для автоматизації багатьох завдань, включаючи розгортання. Jenkins підтримує широкий спектр розширень, які можна використовувати для адаптації його до ваших конкретних потреб.
- Deployer – це ще один інструмент для автоматизації розгортання, який підтримує широкий спектр функцій, включаючи розгортання у контейнерах. Deployer – це хороший вибір для розгортання веб-додатків у складних середовищах.

Донецький національний технічний університет



Тема 5. Система контейнеризації Docker.

Лекція 8. Основи Docker

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

317

- ☐ Мета лекції
- ☐ Основи роботи в Docker Лекційні питання

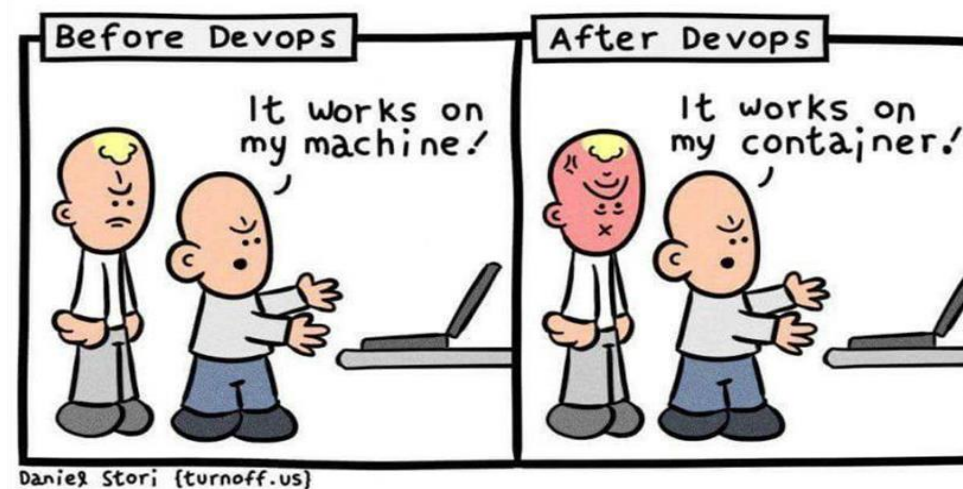


Мета лекції

- Розглянути принципи роботи з контейнерами



Контейнери...





Ще раз про namespaces

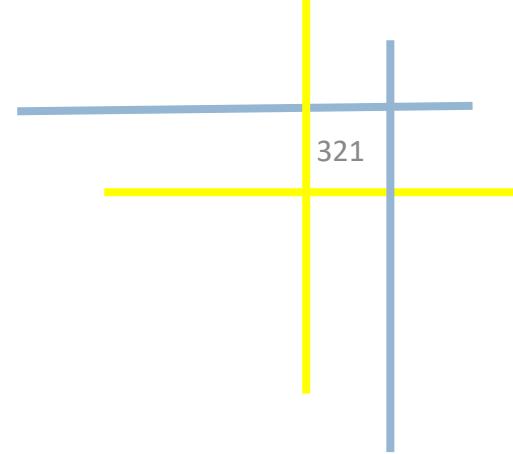
- ☐ User Namespaces – дозволяє пов'язати процес з іншим набором user id
- ☐ Mount namespaces - дозволяє керувати точками монтування
- ☐ PID Namespaces - ізолює ID процесів у системі
- ☐ Network namespaces - ізолює ресурси, пов'язані з мережею
- ☐ Unix Timesharing System NS – дозволяє перевизначати час та ім'я хоста
- ☐ cgroup namespace – ізоляція ресурсів



А що якщо...

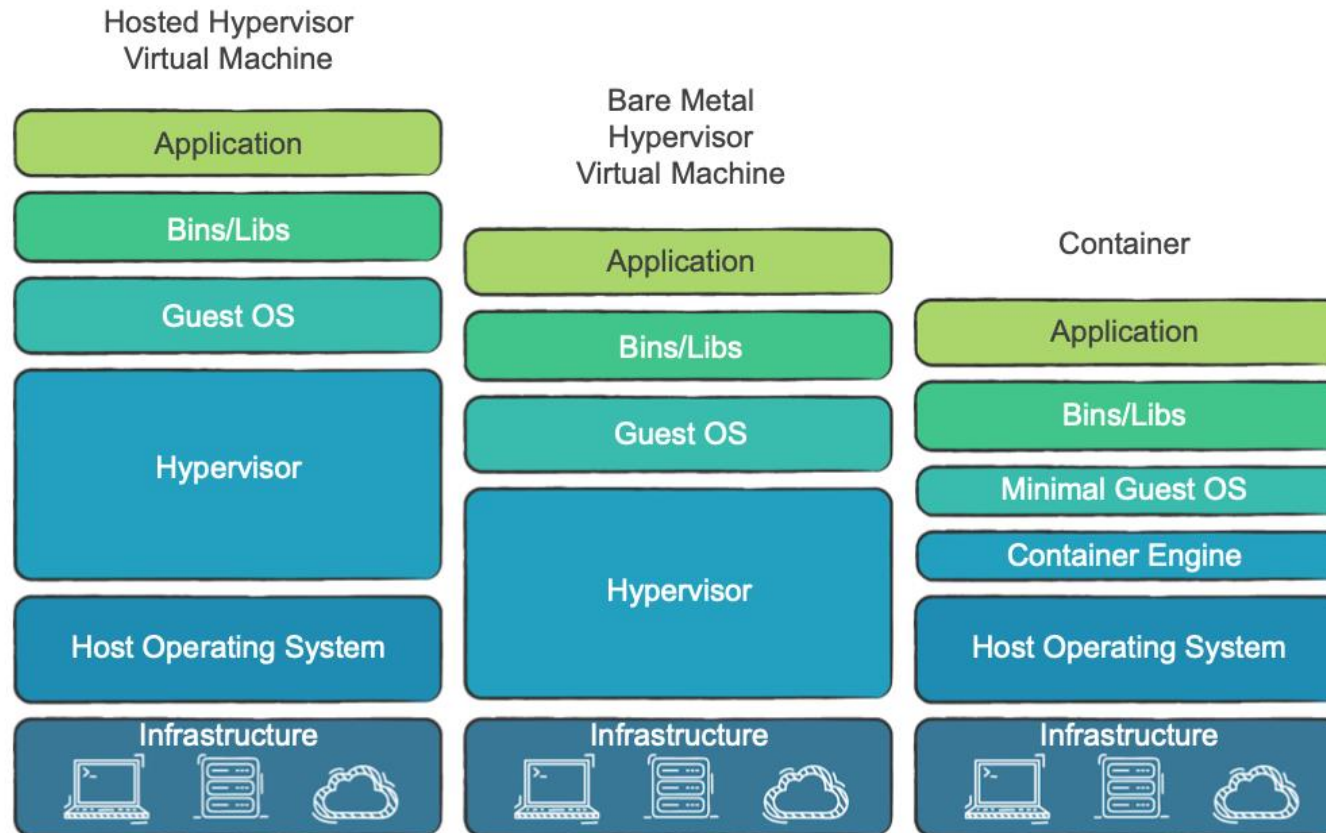
... взяти файли, змонтувати їх за допомогою mount ns та запустити у власному pid , user, uts та cgroups ns?

Вітаю! Ми винайшли докер 😊
(і багато інших систем контейнеризації)





Зв'язок із віртуалізацією



СЛЕРМ



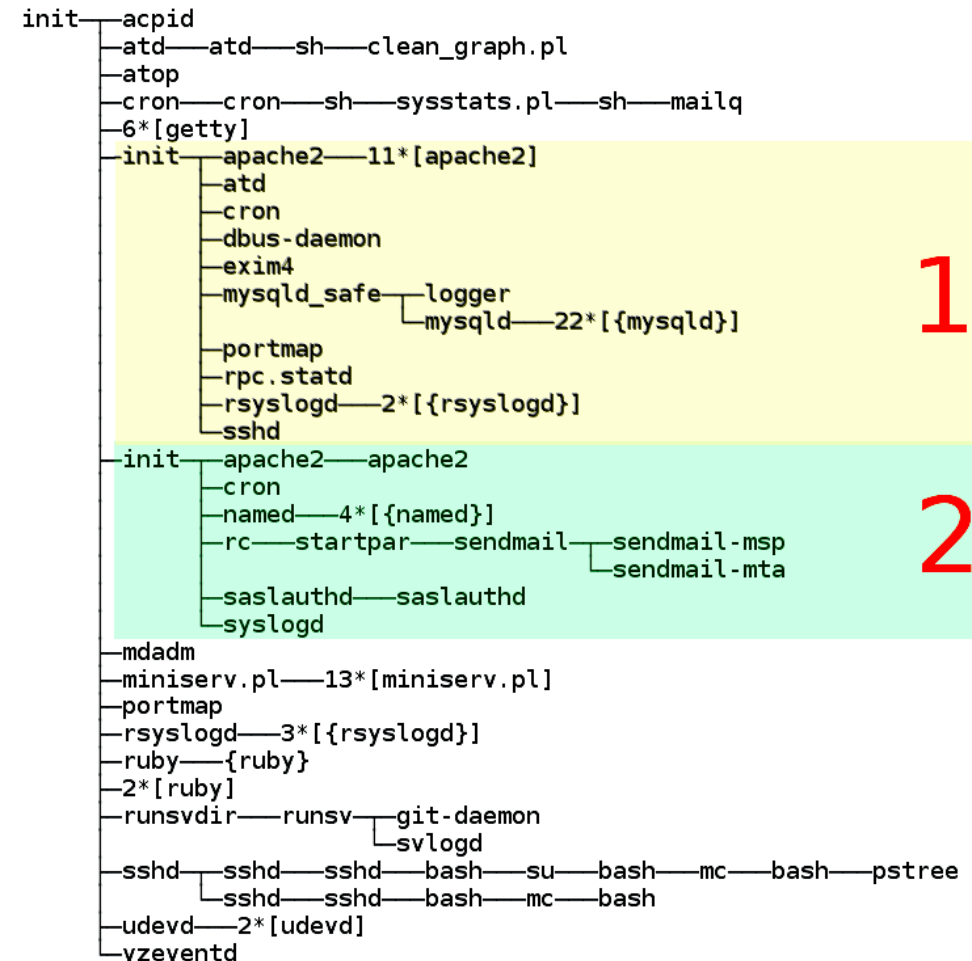
Еволюція контейнеризації

- Unix, 1971
- **chroot**, 1982
- Linux, 17 вересня 1991
- **FreeBSD Jail**, 2003, ще немає контролю ресурсів
- **OpenVZ**, 2005, компанія Parallels, зародження cgroups
- **CGroups**, 2006, раніше “ *process containers*” , компанія Google
- **LXC**, 2008, пізніше **LXC/LXD** стали підтримувати Canonical
- **Docker**, 20 березня 2013, PyCon , компанія dotCloud



OpenVZ

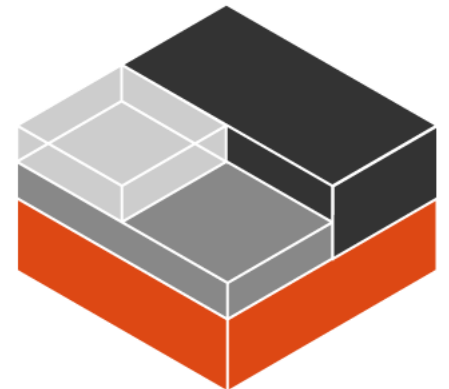
- ❑ Створило концепцію VPS та VE (virtual env)
- ❑ Складається з модифікованого ядра Linux і користувацьких утиліт
- ❑ Є базовою платформою для Virtuozzo – пропрієтарного продукту Parallels, Inc.
- ❑ Є checkpoint та міграції





LXC/LXD

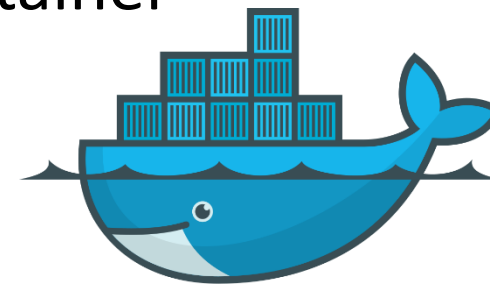
- ☐ LXC – *Linux Containers* - попередник docker
- ☐ LXD – надбудова над LXC, демон для керування
- ☐ Містить у собі систему ініціалізації, та ін.
- ☐ Якщо потрібно запхати legacy- рішення в контейнер
- ☐ Якщо потрібно запустити щось дуже не типове
- ☐ Нині навіть більш гнучкий, ніж Docker





Docker

- ❑ ПЗ для автоматизації розгортання та керування програмами в середовищах з підтримкою контейнеризації
- ❑ Дозволяє «упакувати» застосунок з усім його оточенням та залежностями у контейнер
- ❑ Перша версія базувалася на LXC та Aufs
- ❑ З 2015 року почав використовувати власну бібліотеку, що абстрагує віртуалізаційні можливості ядра Linux - libcontainer



docker



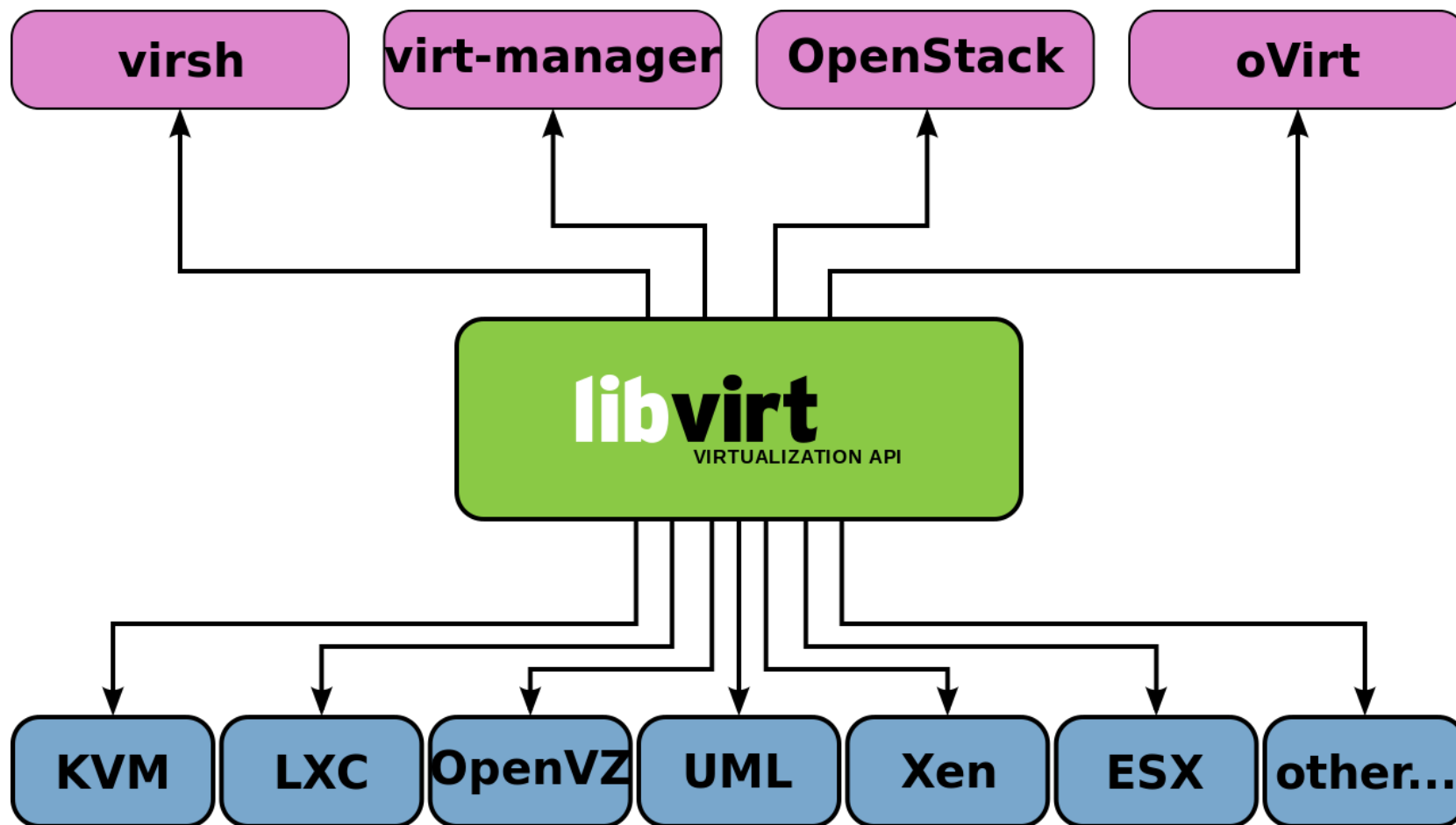
Технологія використовується неправильно, якщо ви...

- ☐ пакуєте всередину docker- контейнера більше однієї програми або запустили кілька **незалежних** процесів
- ☐ ставите всередину docker- контейнера systemd або init
- ☐ змінюєте схему складання логів усередині docker- контейнера з використанням journald або rsyslog
- ☐ накопичуєте зміни всередині docker- контейнера





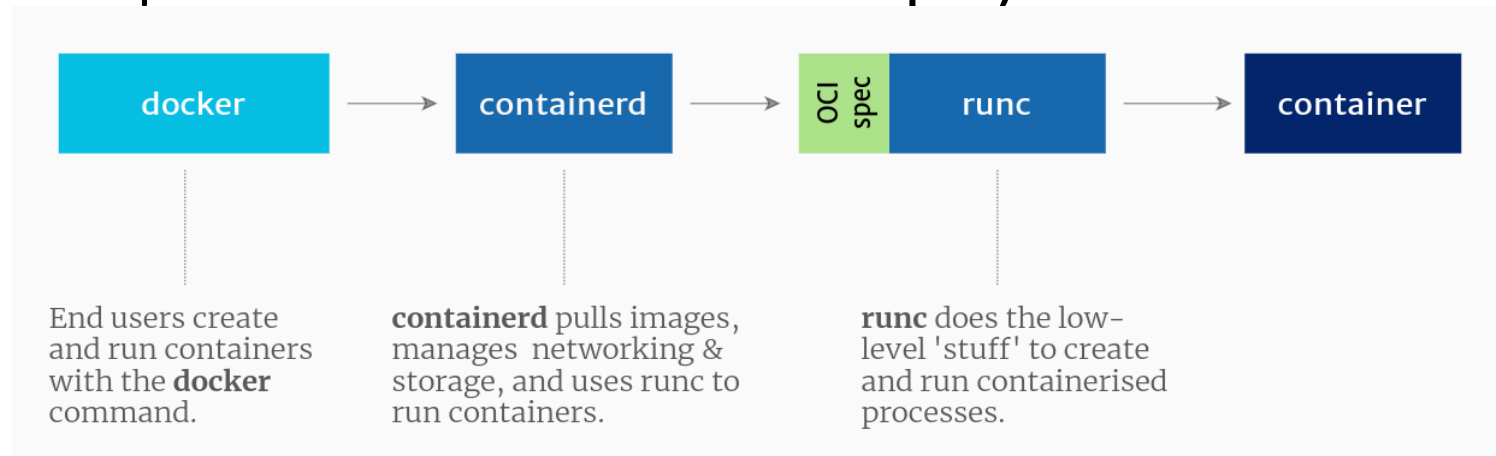
Libvirt - набір інструментів для керування віртуалізацією





OCI - Open Container Initiative

- ❑ Open Container Initiative (OCI) – це проект Linux Foundation, заснований 2015 року компанією Docker, Inc, метою якого є розробка стандартів контейнеризації
- ❑ Не плутати з **Container Runtime Interface (CRI)** , визначає API між Kubernetes та Container Runtime (середовищем виконання контейнерів)





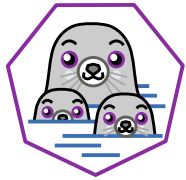
Docker? containerd? runc?

- ☐ Хороша стаття на тему
- ☐ З Docker виділили окремий проект containerd – демон, який управляє контейнерами, образами тощо.
- ☐ Docker тепер є консольною утилітою та GUI до containerd
- ☐ containerd працює з середовищем виконання контейнерів, еталонна реалізація - runc



Podman , Buildah , CRI-O

- ❑ **Podman** - безпосередня взаємодія з контейнерами та сховищем образів через процес runC
- ❑ **Buildah** — збирання та завантаження до реєстру образів
- ❑ **CRI-O** — середовище для систем оркестрації контейнерів .
Тобто, по суті заміна Docker для Kubernetes



podman



buildah



cri-o



Термінологія Docker

- ❑ Образ (image) – шаблон для створення контейнерів
- ❑ Тег образу (tag) – ідентифікатор, який описує стан образу
- ❑ Контейнер (container) – запущений інстанс докер образу
- ❑ Том (volume) – персистентне місце для зберігання даних у контейнері
- ❑ Реєстр/репозиторій образів (registry) – хмарне сховище образів



Docker образ (препаруємо busybox)

Образ – це архів із файлами. Архів складається із:

- ❑ файлу з метаданими `manifest.json`
- ❑ конфіга, що і як запускати (довгий хеш.json)
- ❑ і шарів (папочка з хешом), шар містить усі змінені файли. Шарів може бути багато, перший шар містить усі базові файли операційної системи

```
├── c98db043bed913c5a7b59534cbf8d976122f98b75cb00baabf8af888041e4f9d.json
├── e9589b7a23ce99365f3b7a2823b7510460a1dbe565b49f9147aefdbfb4457742
│   ├── VERSION
│   ├── json
│   └── layer.tar
├── manifest.json
└── repositories
```

```
layer
├── bin
├── dev
├── etc
├── home
├── root
├── tmp
├── usr
└── var
```



OverlayFS

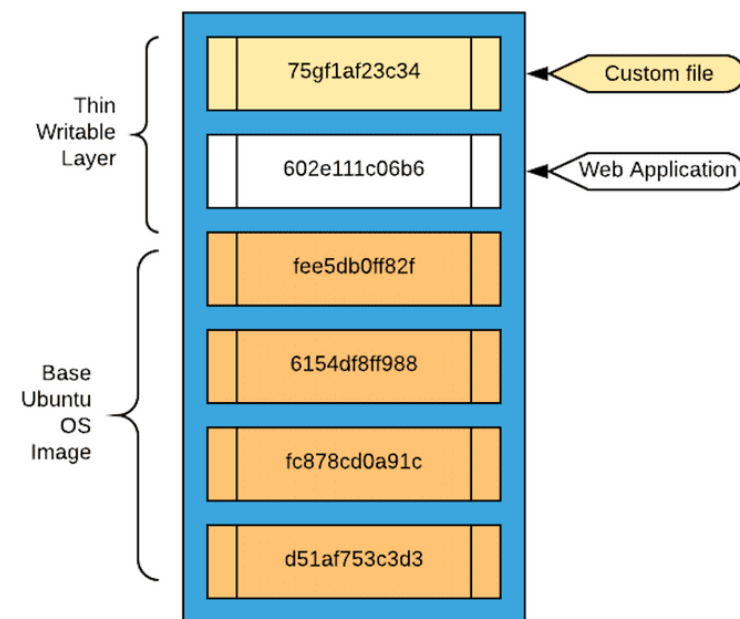
- ❑ **Union mount** - це тип файлової системи, яка створює ілюзію злиття вмісту декількох каталогів в один без зміни вихідних (фізичних) даних в оригінальних джерелах
- ❑ **OverlayFS** - включена в ядро Linux з версії 3.18 (26 жовтня 2014), файлова система, що використовує дефолтний драйвер Docker **overlay2**
- ❑ Завдяки об'єднаній файловій системі, в Docker потрібно створити лише один шар поверх образу, а решта його частини може використовуватися всіма контейнерами



Дивимосся відмінності від образу

- ❑ Як тільки ми внесемо зміни до працюючого контейнера – ми додамо новий шар змін до базового образу, з якого його запустили.
- ❑ Подивимосся перелік змін:
`docker diff < container_id_or_name >`

```
student@debian:~$ docker diff checkintegriy
A /data
A /data/output
C /root
A /root/.bash_history
C /.dockerenv
student@debian:~$
```





Docker образ та registry

Образ можна і потрібно качати з Інтернету. Для завантаження використовується команда **docker pull < ім'я образу >**

Зазвичай образ має назву у наступному форматі:

`<registry>/< ім'я >:<tag>`

Наприклад: *quay.io / coreos / etcd:v3.5.4*

Але можна зробити і **docker pull nginx**, це теж саме, що **docker pull registry.hub.docker.com /library/ nginx:latest**

hub.docker.com - це стандартний репозиторій для всіх образів



Docker push

- ☐ Свій образ можна завантажити назад до Інтернету
`docker push < image_name_or_id >`
- ☐ Попередньо образ можна перетегати за допомогою:
`docker image tag <old> <new>`
- ☐ Створити свій образ можна або збиранням,
або прямо з працюючого контейнера:
`docker container commit < container_id > < new_image_name >`
- ☐ Куди?
hub.docker.com , local docker registry, gitlab -registry,
nexus, ...



Запуск контейнера із образу

docker image ls

docker run -it busybox sh

- ❑ **-it** - прапори, що включають перенаправлення stdin і псевдо tty, дозволяють працювати з контейнером в інтерактивному режимі
- ❑ **busybox** – ім'я образу
- ❑ **sh** – команда для запуску у контейнері

```
[/ # hostname
6a12f40a17d6
[/ # ps -a
PID    USER      TIME  COMMAND
   1   root         0:00  sh
   8   root         0:00  ps -a
[/ # ls -la /
total 44
drwxr-xr-x  1 root    root      4096 Sep 11 19:56 .
drwxr-xr-x  1 root    root      4096 Sep 11 19:56 ..
-rwxr-xr-x  1 root    root         0 Sep 11 19:56 .dockerenv
drwxr-xr-x  2 root    root     12288 Sep  1 16:42 bin
drwxr-xr-x  5 root    root      340 Sep 11 19:56 dev
drwxr-xr-x  1 root    root      4096 Sep 11 19:56 etc
drwxr-xr-x  2 nobody  nobody    4096 Sep  1 16:42 home
dr-xr-xr-x 179 root    root         0 Sep 11 19:56 proc
drwx----- 1 root    root      4096 Sep 11 19:56 root
dr-xr-xr-x 13 root    root         0 Sep 11 19:56 sys
drwxrwxrwt  2 root    root      4096 Sep  1 16:42 tmp
drwxr-xr-x  3 root    root      4096 Sep  1 16:42 usr
drwxr-xr-x  4 root    root      4096 Sep  1 16:42 var
[/ #
```



Основні команди

- ❑ **docker ps** – список запущених контейнерів
- ❑ **docker attach <id або ім'я контейнера >** - підключитися до контейнера в інтерактивному режимі
- ❑ **docker exec <id або ім'я контейнера > < команда >** - виконати команду в робочому контейнері
- ❑ **docker stop <id або ім'я контейнера >** - зупинити робочий контейнер
- ❑ **docker kill <id або ім'я контейнера >** - примусово зупинити контейнер
- ❑ **docker start <id або ім'я контейнера >** - повторно запустити зупинений контейнер
- ❑ **docker rm <id або ім'я контейнера >** - вида

```
[sh-3.2$ docker run -d --name=hello busybox sleep 600
7c6f587447fd12f0b061657fecb4de74b0a6384c029011b99d0c3d41a3262677
[sh-3.2$ docker stop hello
hello
[sh-3.2$ docker start hello
hello
[sh-3.2$ docker rm -f hello
hello
```



Docker та дані

За замовчуванням дані в контейнері не зберігаються між запусками !

```
docker run -it -e \  
"MYSQL_ALLOW_EMPTY_ROOT_PASSWORD=1" mysql:5.7
```

Щоб це виправити, використовуються volume

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
4 rows in set (0.00 sec)  
  
mysql> create database test;  
Query OK, 1 row affected (0.00 sec)  
  
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
| test |  
+-----+  
5 rows in set (0.00 sec)  
  
mysql> sh-3.2$  
sh-3.2$ docker exec -it 7d8beed2fdd4 mysql  
Welcome to the MySQL monitor.  Commands end with ; or \g.  
Your MySQL connection id is 2  
Server version: 5.7.37 MySQL Community Server (GPL)  
  
Copyright (c) 2000, 2022, Oracle and/or its affiliates.  
  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
  
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sys |  
+-----+  
4 rows in set (0.00 sec)
```




Типи volume

При запуску контейнера підмонтувати volume можна за допомогою опції – **v**

Існує кілька типів volume:

- ☐ bind – прокидає у контейнер директорію з хоста
- ☐ volume для докера – сховище всередині докеру
- ☐ tmpfs – створює том в оперативній пам'яті, не зберігає дані, але зручно для тимчасових даних з великим іорс



Bind volume

Bind дозволяє прокинути директорію на хості в контейнер

Вказуємо `-v < шлях на хості >:< шлях у контейнері >`
Так, так можна прокинути цілий device , docker- сокет або навіть усю ієрархію фс хоста. **Але так не можна робити.**

```
[sh-3.2$  
[sh-3.2$ ls  
[sh-3.2$ docker run --rm busybox touch /data/hello_bind  
touch: /data/hello_bind: No such file or directory  
[sh-3.2$ ls  
[sh-3.2$ docker run --rm -v $(pwd):/data busybox touch /data/hello_bind  
[sh-3.2$ ls  
hello_bind  
[sh-3.2$
```



Стандартний docker volume

Докер дозволяє створювати іменоване сховище даних за допомогою команди

docker volume create <ім'я volume>

У прикладі нижче, **docker volume create hello**

```
[sh-3.2$ docker run --rm -it -v hello:/data busybox sh
[/ # echo "hello volume!" > /data/message
[/ #
sh-3.2$ docker run --rm -it -v hello:/data busybox cat /data/message
hello volume!
sh-3.2$
```



Docker та мережа

Зрозуміло, у docker можна запускати мережеві програми, такі як веб-сервера.

Наприклад, **docker run --rm nginx**

Програми будуть доступні всередині докеру мережі, але щоб отримати до них доступ з комп'ютера, необхідно прокинути порт. Це робиться за допомогою опції

–p < порт на хості куди прокинути >:< який порт із контейнера прокинути >

Наприклад, **docker run --rm -p 8080:80 nginx**

Крім того, для вбудованого “service discovery” у docker існує локальний **DNS** (127.0.0.11:53)

Для параметра **name** контейнера автоматично настраюється перетворення імені на адресу контейнера всередині віртуальної мережі

```
sh-3.2$ curl -I localhost:8080
HTTP/1.1 200 OK
Server: nginx/1.23.1
Date: Sun, 11 Sep 2022 20:57:01 GMT
Content-Type: text/html
Content-Length: 615
Last-Modified: Tue, 19 Jul 2022 14:05:27 GMT
Connection: keep-alive
ETag: "62d6ba27-267"
Accept-Ranges: bytes
```



Створення свого образу

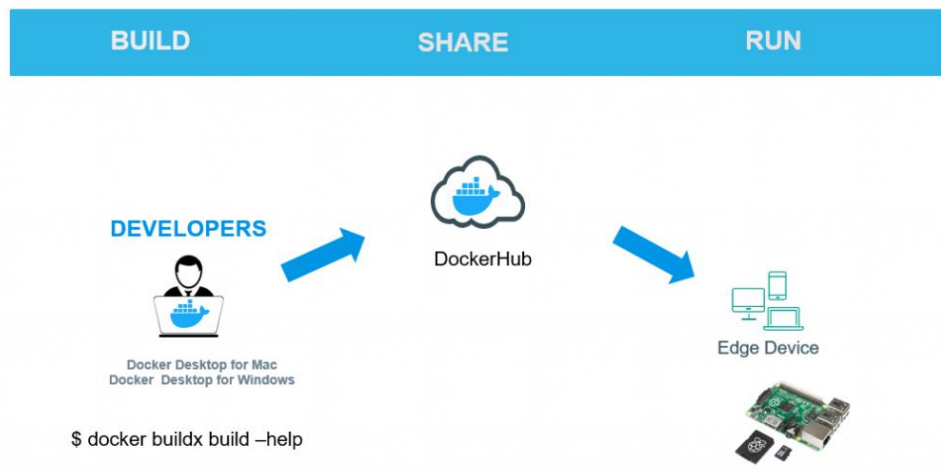
Для складання свого образу використовується команда **docker build** та особливий файл **Dockefile** , який містить набір інструкцій, що описують шари в образі.

Для складання образу використовується контекст, вказаний у команді `docker build` , весь вміст завантажується в докер демон, якщо якісь файли не потрібні, їх можна оновити у файл **.dockerignore**



buildx

- ❑ Складання під інші платформи, наприклад ви працюєте на маку, а використовувати його плануєте на Raspberry Pi
- ❑ `docker buildx build --platform linux/arm64 -f Dockerfile.alpine -t devdotnetorg/alpine-ssh:aarch64 . --load`



docker/buildx

Docker CLI plugin for extended build capabilities with BuildKit



60

Contributors

99

Used by

20

Discussions

2k

Stars

300

Forks

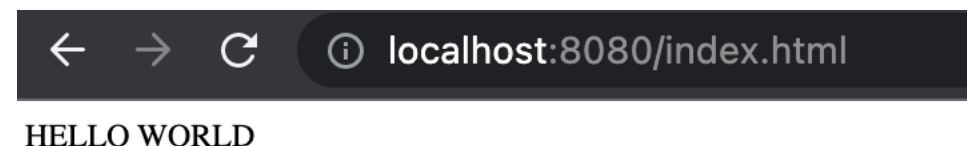




Основи Dockerfile

- ❑ Завжди треба вказувати базовий образ за допомогою **FROM**. Найбільш базовий образ **scratch**
- ❑ Щоб скопіювати файл із контексту **COPY** /щось у контексті /кудись в образі
- ❑ Щоб запустити команду в образі **RUN** команда
- ❑ **ENV** дозволяє визначати змінні оточення для майбутнього контейнера

```
index.html x
index.html > h1
1  <h1>HELLO WORLD</h1>
2
3
Dockerfile x
Dockerfile > ...
1  FROM nginx
2
3  COPY index.html /usr/share/nginx/html/
```





Використання шарів

Команди RUN і COPY породжують шари, шар - це зміни у файловій системі, викликані вищою командою (нові файли або видалення)
ВАЖЛИВО! Шар «запікається» під час виклику команди.

Таким чином, у кожному шарі рекомендується мінімізувати кількість створюваних непотрібних файлів, тобто відразу їх чистити.
Наприклад, *RUN yum install package && yum clean all*

```
Dockerfile > ...  
1 FROM alpine  
2 RUN truncate -s 10M big_file.txt  
3 RUN rm -rf big_file.txt
```




Шари та кеш

Docker вираховує хеш від кожного шару, при спробі його зібрати, хеш шару розраховується в залежності від хеша попереднього шару, тобто, якщо змінився попередній шар, всі наступні шари перезберуться.

```
Dockerfile > ...
1 FROM nginx
2
3 RUN sleep 10
4 COPY index.html /usr/share/nginx/html/

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

$ time docker build -t nginx_cache
[+] Building 10.7s (8/8) FINISHED

=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 106B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:latest
=> CACHED [1/3] FROM docker.io/library/nginx
=> [internal] load build context
=> => transferring context: 59B
=> [2/3] RUN sleep 10
=> [3/3] COPY index.html /usr/share/nginx/html/
=> exporting to image
=> => exporting layers
=> => writing image sha256:db653b8e135bd443f53aafb6d6c8a6e5d6add482f015aceeafb6261826ebb01
=> => naming to docker.io/library/nginx_cache

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

real    0m11.657s
user    0m0.143s
cpu     0mA 78Rre

$ time docker build -t nginx_cache
[+] Building 0.1s (0/0) FINISHED

=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 36B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/nginx:latest
=> [1/3] FROM docker.io/library/nginx
=> [internal] load build context
=> => transferring context: 31B
=> CACHED [2/3] RUN sleep 10
=> CACHED [3/3] COPY index.html /usr/share/nginx/html/
=> exporting to image
=> => exporting layers
=> => writing image sha256:db653b8e135bd443f53aafb6d6c8a6e5d6add482f015aceeafb6261826ebb01
=> => naming to docker.io/library/nginx_cache

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

real    0m0.480s
user    0m0.115s
sys     0m0.069s
```



Чимало шарів з кешем або мінімізувати розмір?!

- ☐ Як і скрізь в ІТ немає ідеальної відповіді на це питання, все залежить від умов використання образу та завдань:
- ☐ Образ часто скачується, але мало змінюється – мінімізуйте розмір, зменшивши кількість шарів.
- ☐ Образ постійно перезбирається – виділіть шари, які рідко змінюються і додавайте зміни в останньому шарі (наприклад, спочатку встановлення залежностей, потім копіюємо вихідний код)



Лайфхак з компілюваними програмами

- ❑ Докер дозволяє під час збирання використовувати додаткові образи для збирання коду, відокремивши складальну частину та експлуатаційну, з метою мінімізації місця.

```
# syntax=docker/dockerfile:1
FROM golang:1.16
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go ./
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app ./
CMD ["/app"]
```

Донецький національний технічний університет



Тема 6. Системи управління конфігураціями.

Лекція 9. Система управління конфігураціями Chef

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

353

- ☐ Вступ
- ☐ Що таке Chef?
- ☐ Основні концепції Chef
- ☐ Інструментарій Chef
- ☐ Практичні приклади
- ☐ Висновок



Що таке Chef?

- ☐ Chef – це інструмент автоматизації конфігурації, який використовується для управління інфраструктурою ІТ-систем.
- ☐ Chef дозволяє автоматизувати завдання, такі як:
 - Установка та налаштування програмного забезпечення
 - Управління конфігураціями систем
 - Моніторинг систем





Роль Chef у сучасних інфраструктурних рішеннях

- ❑ Chef стає все більш популярним інструментом для управління інфраструктурою в сучасних організаціях.
- ❑ Це пов'язано з наступними факторами:
 - Зростання складності інфраструктури
 - Необхідність забезпечення гнучкості та масштабованості
 - Зменшення витрат на управління інфраструктурою



Переваги Chef

Загальні переваги використання Chef:

- ☐ **Покращена точність і надійність конфігурацій:** Chef дозволяє гарантувати, що всі системи будуть налаштовані однаково.
- ☐ **Зменшення часу та зусиль, необхідних для управління інфраструктурою:** Chef може автоматизувати багато завдань, які зазвичай виконуються вручну.
- ☐ **Покращена безпека інфраструктури:** Chef може допомогти виявити і усунути потенційні вразливості в конфігураціях систем.



Інфраструктура як код

- ❑ Інфраструктура як код (IaC) – це концепція, при якій інфраструктура описується як код, який можна зберігати в репозиторії та керувати ним за допомогою інструментів управління конфігурацією.
- ❑ IaC має ряд переваг, таких як:
 - **Покращена керованість:** інфраструктуру можна легко змінювати та керувати нею.
 - **Покращена повторюваність:** інфраструктуру можна легко створювати та налаштовувати на різних системах.
 - **Покращена прозорість:** інфраструктуру можна легко розуміти та контролювати.



Декларативний підхід

- ❑ Декларативний підхід – це метод, при якому інфраструктура описується в термінах того, що потрібно зробити, а не того, як це зробити.
- ❑ При використанні дедуктивного підходу:
 - Інфраструктура описується в термінах бажаного стану.
 - Інструмент управління конфігурацією відповідає за те, щоб інфраструктура прийшла в бажаний стан.

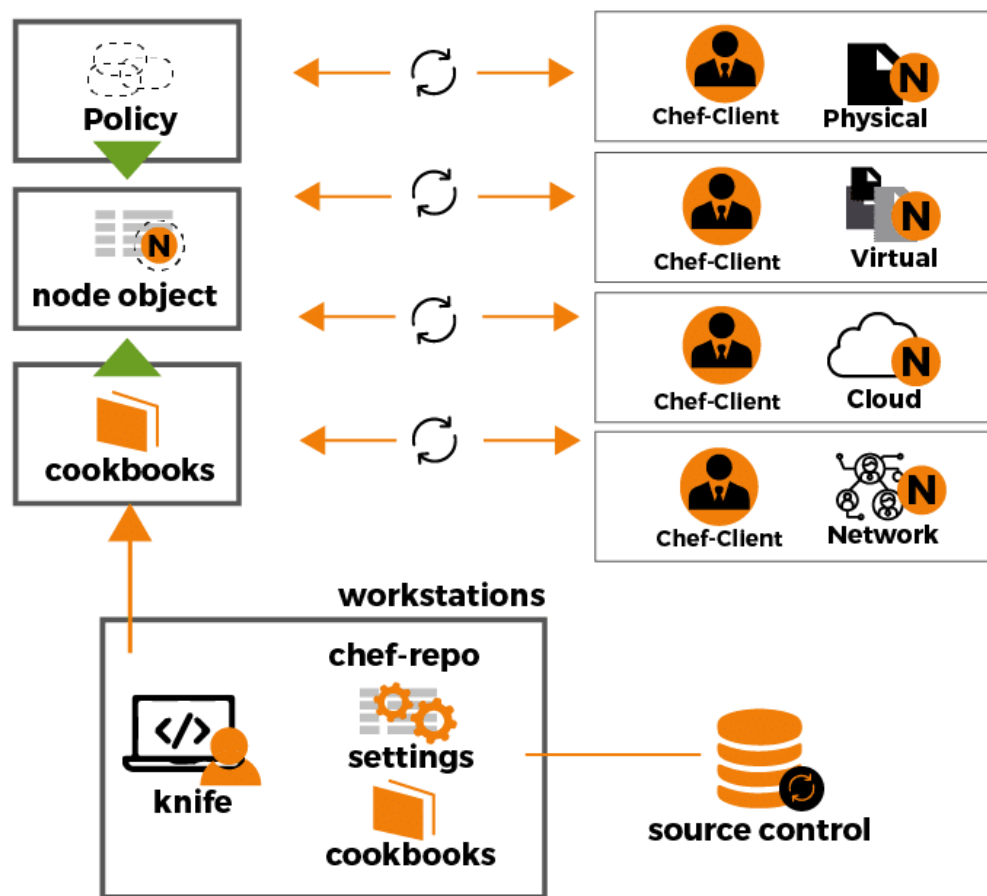


Автоматизація конфігурації

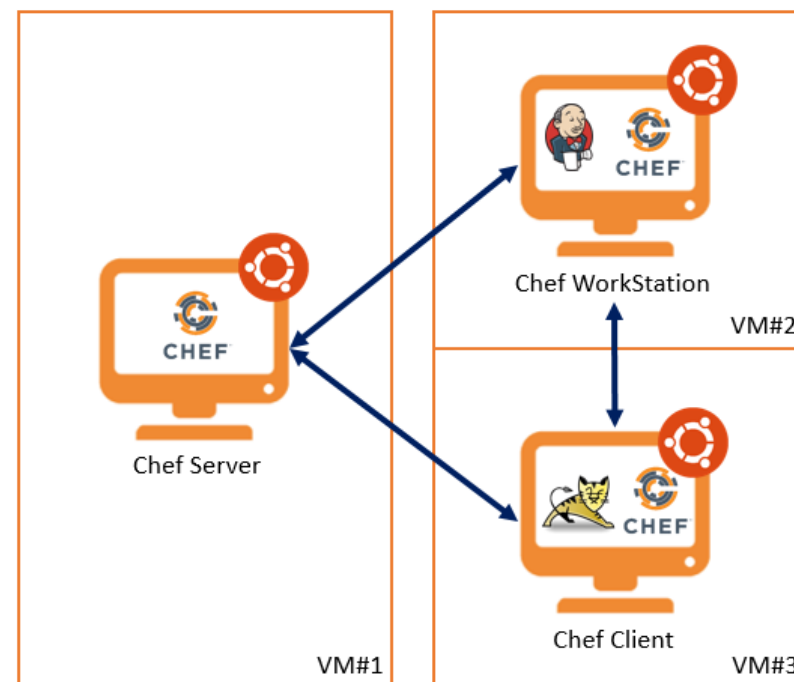
- ❑ Автоматизація конфігурації – це процес налаштування інфраструктури автоматично за допомогою інструментів управління конфігурацією.
- ❑ Автоматизація конфігурації має ряд переваг, таких як:
 - **Покращена точність:** інфраструктура буде налаштована однаково на всіх системах.
 - **Зменшення часу та зусиль, необхідних для управління інфраструктурою:** конфігурація може бути змінена швидко та легко.
 - **Покращена безпека:** інфраструктура може бути захищена від помилок та вразливостей.



Архітектура Chef



Chef Architecture





Chef Workstation

- ☐ Chef Workstation – це комп'ютер, на якому розробляються та тестуються рецепти Chef.
- ☐ Chef Workstation використовується для наступних завдань:
 - Створення та редагування рецептів Chef.
 - Тестування рецептів Chef.
 - Надсилання рецептів Chef на Chef Server.

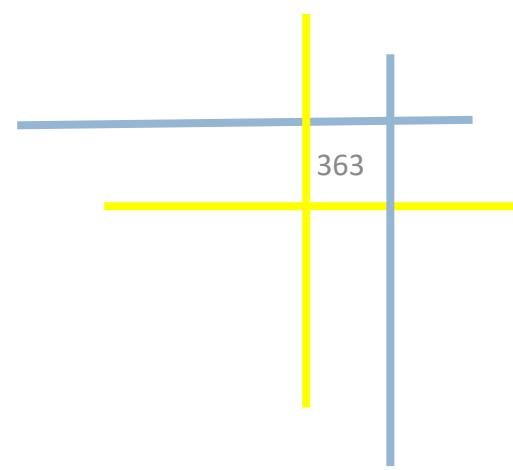


Chef Server

- ☐ Chef Server – це сервер, який керує виконанням рецептів Chef.
- ☐ Chef Server використовується для наступних завдань:
 - Зберігання рецептів Chef.
 - Розподіл рецептів Chef на Chef Clients.
 - Контроль виконання рецептів Chef на Chef Clients.



Chef Client



- ☐ Chef Client – це агент, який виконує рецепти Chef на системах.
- ☐ Chef Client використовується для наступних завдань:
 - Отримання рецептів Chef від Chef Server.
 - Виконання рецептів Chef на системі.
 - Повідомлення Chef Server про результат виконання рецептів.

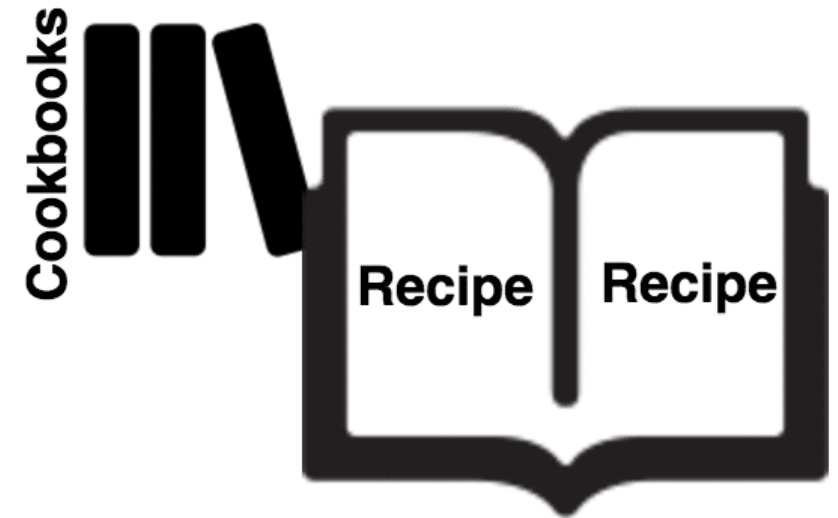


Chef Client



Cookbooks

- ❑ Cookbooks – це колекція рецептів, які використовуються для налаштування систем.
- ❑ Cookbooks організовані за темами, наприклад, "Web Servers", "Databases" або "Applications".
- ❑ Cookbooks написані на мові Ruby.





Recipes

- ☐ Recipes – це послідовність команд, які виконуються для налаштування системи.
- ☐ Recipes описують, як налаштувати систему в певному стані.
- ☐ Recipes написані на мові Ruby.



Запуск конфігурації

- ☐ Запуск конфігурації – це процес, за допомогою якого Chef налаштовує систему.
- ☐ Запуск конфігурації починається з того, що Chef Client отримує рецепти від Chef Server.
- ☐ Chef Client потім виконує рецепти на системі, щоб налаштувати її в бажаний стан.



Операційний цикл

- ☐ Операційний цикл – це процес, за допомогою якого Chef підтримує систему в бажаному стані.
- ☐ Операційний цикл починається з того, що Chef Client перевіряє статус системи.
- ☐ Якщо статус системи не відповідає бажаному, Chef Client виконує необхідні зміни.



Декларативні описи ресурсів

- ☐ Декларативні описи ресурсів – це спосіб опису того, як повинна бути налаштована система.
- ☐ Декларативні описи ресурсів описують бажаний стан системи, а не те, як його досягти.



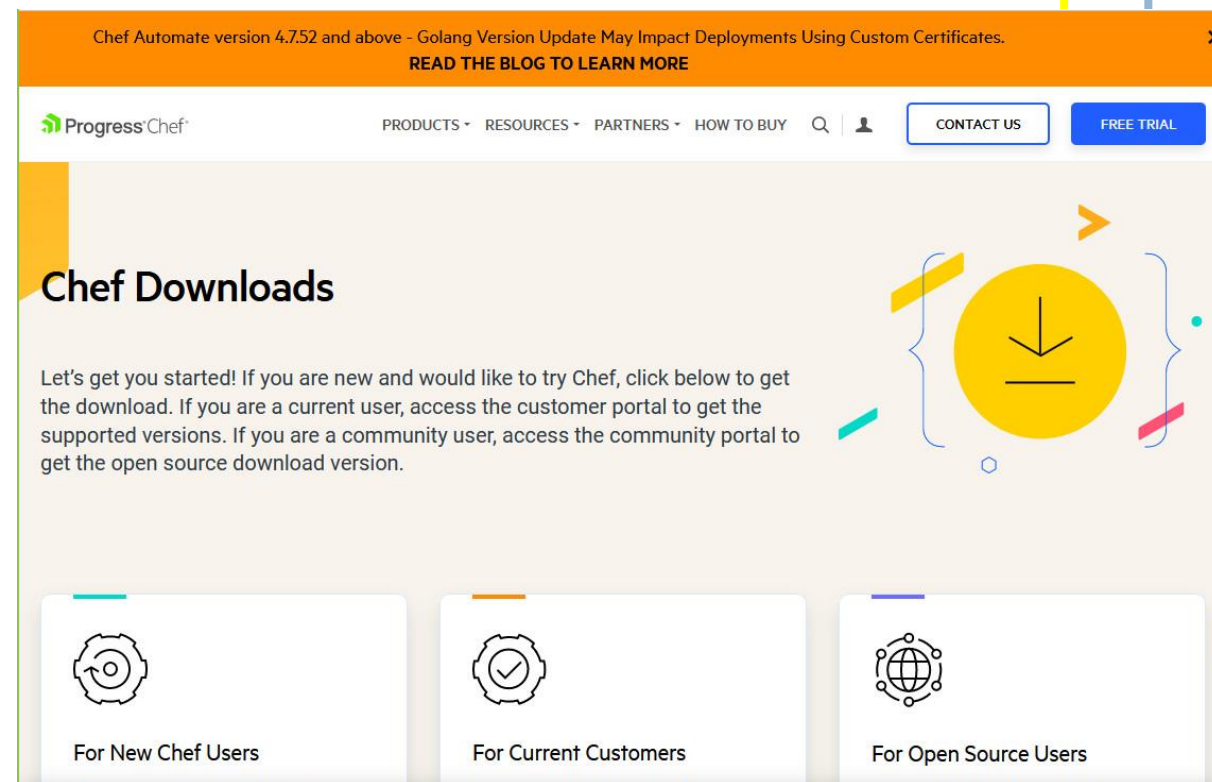
Зберігання стану конфігурації

- ☐ Chef зберігає стан конфігурації системи, щоб можна було відстежувати зміни.
- ☐ Стан конфігурації зберігається в базі даних Chef.



Інсталяція та конфігурація Chef Workstation

- ❑ Щоб встановити Chef Workstation, потрібно:
- Завантажити інсталяційний пакет з веб-сайту Chef (<https://www.chef.io/downloads>).
 - Виконати інсталяційний пакет.
 - Створити обліковий запис Chef.





Створення та управління Cookbooks

❑ Щоб створити Cookbooks, потрібно:

- Створити каталог для зберігання Cookbooks.
- Створити файл з розширенням .rb для кожного рецепта.
- Відредагувати файли рецептів, щоб описати, як налаштувати систему.



Налаштування та управління Chef Server

372

❑ Щоб налаштувати Chef Server, потрібно:

- Встановити Chef Server на комп'ютер.
- Створити базу даних для Chef Server.
- Налаштувати параметри Chef Server.



Інтеграція Chef з іншими інструментами

- ☐ Chef можна інтегрувати з іншими інструментами управління інфраструктурою, такими як Ansible, Puppet або SaltStack.
- ☐ Інтеграція Chef з іншими інструментами може допомогти організаціям покращити ефективність та масштабованість автоматизації.



Структура Cookbooks

- ☐ Cookbooks організовані в каталоги за темами або іншими критеріями.
- ☐ Кожен Cookbook містить один або кілька Recipes.
- ☐ Recipes – це послідовність команд, які виконуються для налаштування системи.



Типові схеми організації Cookbook

- ❑ Cookbooks можна організувати в каталоги за будь-якою схемою, яка відповідає потребам організації.
- ❑ Типові схеми організації Cookbooks включають:
 - **За темами:** Cookbook `webserver` може містити Recipes для установки, налаштування та запуску різних веб-серверів.
 - **За операційними системами:** Cookbook `linux_webserver` може містити Recipes для установки, налаштування та запуску веб-сервера на Linux.
 - **За етапами розгортання:** Cookbook `webserver_install` може містити Recipes для установки веб-сервера, Cookbook `webserver_configure` може містити Recipes для налаштування веб-сервера, а Cookbook `webserver_start` може містити Recipes для запуску веб-сервера.



Способи створення Cookbook

- ❑ Cookbooks можна створювати вручну або за допомогою інструментів, таких як ChefDK.
 - ChefDK – це комплект інструментів, який включає все необхідне для створення, тестування та розгортання Cookbooks.
- ❑ Cookbooks можна публікувати в репозиторіях, таких як Supermarket або Chef Community Cookbooks.
 - Supermarket – це загальнодоступний репозиторій Cookbooks, який можна використовувати для пошуку та завантаження Cookbooks, створених іншими користувачами.
 - Chef Community Cookbooks – це репозиторій Cookbooks, який підтримується спільнотою Chef.



Приклад Cookbook₁

❑ Цей Cookbooks містить три Recipes:

- `webserver::install` встановлює веб-сервер Apache.
- `webserver::configure` налаштовує конфігурацію веб-сервера Apache.
- `webserver::start` запускає веб-сервер Apache.

❑ Кожен Recipe містить послідовність команд, які виконуються для виконання певного завдання. У цьому випадку Recipes використовують ресурси Chef для установки пакетів, налаштування файлів та запуску служб.

❑ Щоб використовувати цей Cookbooks, потрібно створити файл `metadata.rb`, який містить метадані про Cookbooks. Файл `metadata.rb` повинен містити такі поля:

- `name`: Ім'я Cookbooks.
- `maintainer`: Ім'я та контактна інформація розробника Cookbooks.
- `version`: Версія Cookbooks.

```
# Cookbook for installing and configuring the Apache web server

name 'webserver'
maintainer 'Your Name'
version '1.0.0'

# Recipes

recipe 'webserver::install' do
  # Install the Apache package
  package 'apache2'
end

recipe 'webserver::configure' do
  # Configure the Apache configuration file
  file '/etc/apache2/conf/httpd.conf' do
    content 'ServerName localhost'
  end
end

recipe 'webserver::start' do
  # Start the Apache service
  service 'apache2' do
    action :start
  end
end
```



Приклад Cookbook₂

❑ Ось приклад файлу `metadata.rb` для цього Cookbooks:

```
name 'webserver'  
maintainer 'Your Name'  
version '1.0.0'
```

❑ Після створення файлу `metadata.rb` можна розгорнути Cookbooks на системах Linux, використовуючи інструменти Chef, такі як Chef Client або Chef Server.

❑ Ось приклад команди, яку можна використовувати для розгортання Cookbooks на системі Linux:

```
chef-client -o webserver
```

❑ Ця команда розгорне всі Recipes в Cookbooks webserver на системі.



Опис ресурсів в Recipes₁

- ☐ Recipes використовують ресурси для налаштування системи.
- ☐ Ресурси описують, що потрібно зробити для налаштування системи.
- ☐ Ресурси можуть виконувати різні завдання, такі як:
 - Встановлення пакетів.
 - Налаштування файлів.
 - Створення служб.



Опис ресурсів в Recipes₂

- ☐ Усі ресурси Chef мають загальну структуру опису:
 - # Назва ресурсу
 - # Атрибути ресурсу
 - # Дії ресурсу
- ☐ Назва ресурсу визначає тип ресурсу, який потрібно виконати. У наведених далі прикладах ресурси мають назви `package`, `file` та `service`.
- ☐ Атрибути ресурсу використовуються для налаштування поведінки ресурсу. У наведених далі прикладах атрибут `content` ресурсу `file` використовується для установки значення вмісту файлу конфігурації веб-сервера Apache.
- ☐ Дії ресурсу визначають, що потрібно зробити з ресурсом. У наведених далі прикладах дія `action` ресурсу `service` використовується для запуску служби веб-сервера Apache.
- ☐ Існує багато інших ресурсів Chef, які можна використовувати для виконання різних завдань з управління інфраструктурою. Повний список ресурсів Chef можна знайти в документації Chef.



Приклад Recipes₁

□ Приклад опису ресурсу `package` в Recipe:

```
# Install the Apache package  
package 'apache2'
```

Цей код встановлює пакет `apache2` на системі.

□ Ще один приклад опису ресурсу `file` в Recipe:

```
# Configure the Apache configuration file  
file '/etc/apache2/conf/httpd.conf' do  
  content 'ServerName localhost'  
end
```

Цей код налаштовує файл конфігурації веб-сервера Apache, щоб він прослуховував `localhost`.



Приклад Recipes₂

❑ Ще один приклад опису ресурсу `service` в Recipe:

```
# Start the Apache service
service 'apache2' do
  action :start
end
```

Цей код запускає службу веб-сервера Apache.



Використання атрибутів₁

- ☐ Атрибути можна використовувати для налаштування поведінки ресурсів.
- ☐ Атрибути визначаються як ключ-значення.
- ☐ Атрибути можуть бути задані глобально для Cookbook або локально для Recipe.



Використання атрибутів₂

- ❑ Атрибути можна використовувати для налаштування різних аспектів поведінки ресурсів Chef. Повний список атрибутів для кожного ресурсу можна знайти в документації Chef.
- ❑ Ось кілька додаткових прикладів використання атрибутів:
 - Визначення імені пакета або служби для встановлення або запуску.
 - Визначення шляху до файлу або каталогу для налаштування.
 - Визначення значень для змінних або параметрів.
 - Визначення режиму запуску служби або процесу.
 - Визначення прав доступу до файлу або каталогу.
 - Визначення інших атрибутів, які визначаються для конкретного ресурсу.
- ❑ Атрибути можуть бути використані для створення більш гнучких і масштабованих Cookbooks, які можна використовувати для налаштування систем у різних середовищах.



Приклади використання атрибутів₁

□ Приклад 1

```
# Install the Apache package
package 'apache2' do
  #Set the version of the Apache package to install
  version '2.4.41'
end
```

У цьому прикладі атрибут `version` використовується для установки версії пакета Apache, який потрібно встановити.

Приклад 2

```
# Configure the Apache configuration file
file '/etc/apache2/conf/httpd.conf' do
  # Set the ServerName value in the Apache configuration file
  content 'ServerName myserver.example.com'
end
```

У цьому прикладі атрибут `content` використовується для установки значення вмісту файлу конфігурації веб-сервера Apache.



Приклади використання атрибутів₂

□ Приклад 3

```
# Start the Apache service
service 'apache2' do
  # Set the start mode of the Apache service
  start_mode 'manual'
End
```

У цьому прикладі атрибут `start_mode` використовується для установки режиму запуску служби веб-сервера Apache.

□ Приклад 4

```
# Create a user account
user 'myuser' do
  # Set the password for the user account
  password 'mypassword'
end
```

У цьому прикладі атрибут `password` використовується для установки пароля для облікового запису користувача.



Збірка Cookbooks

- ❑ Збірка Cookbooks – це процес створення готового до розгортання пакета з Cookbooks.
- ❑ Збірка Cookbooks виконується за допомогою інструментів Chef, таких як ChefDK або knife.
- ❑ Збірка Cookbooks включає в себе такі етапи:
 - Оновлення метаданих Cookbooks.
 - Використання ресурсів Chef для виконання завдань налаштування.
 - Створення готового до розгортання пакета.



Приклади збірки Cookbook₁

❑ Збірка Cookbooks за допомогою інструменту `knife`
`knife cookbooks build` .

Ця команда збере всі Cookbooks у поточному каталозі.

❑ Збірка Cookbooks за допомогою інструменту `Rake`
`rake build`



Приклади збірки Cookbook₂

❑ Збірка Cookbooks за допомогою власного скрипта

```
# Збірка Cookbooks за допомогою власного скрипта
```

```
# Визначте список Cookbooks, які потрібно зібрати  
cookbooks = ["mys cookbook1", "mys cookbook2"]
```

```
# Для кожного Cookbooks виконайте такі дії:
```

```
for cookbook in cookbooks:
```

```
    # Переконайтеся, що Cookbooks існує
```

```
    if not os.path.isdir(cookbook):
```

```
        raise ValueError("Cookbooks '%s' не існує" % cookbook)
```

```
# Переконайтеся, що Cookbooks містить файл metadata.rb
```

```
if not os.path.isfile(os.path.join(cookbook, "metadata.rb")):
```

```
    raise ValueError("Cookbooks '%s' не містить файл metadata.rb" % cookbook)
```

```
# Зберіть Cookbooks
```

```
knife cookbook build % cookbook
```



Завантаження конфігурації на Chef Server

- ☐ Завантаження конфігурації на Chef Server – це процес розміщення Cookbooks та інших файлів конфігурації на Chef Server.
- ☐ Завантаження конфігурації на Chef Server виконується за допомогою інструментів Chef, таких як ChefDK або `knife`.
- ☐ Завантаження конфігурації на Chef Server включає в себе такі етапи:
 - Імпорт Cookbooks на Chef Server.
 - Завантаження інших файлів конфігурації на Chef Server.



Приклади завантаження конфігурації на Chef Server₁

- ❑ Завантаження Cookbooks на Chef Server за допомогою інструменту `knife`

`knife cookbooks upload` .

Ця команда імпортує всі Cookbooks у поточному каталозі на Chef Server.

- ❑ Завантаження Cookbooks на Chef Server за допомогою інструменту `Rake`

`rake upload`

Ця команда імпортує всі Cookbooks у каталозі `cookbooks` на Chef Server.



Приклади завантаження конфігурації на Chef Server₂

- ❑ Завантаження Cookbooks на Chef Server за допомогою власного скрипта

```
cookbooks = ["mycookbook1", "mycookbook2"]
for cookbook in cookbooks:
    if not os.path.isdir(cookbook):
        raise ValueError("Cookbooks '%s' не існує" % cookbook)
    if not os.path.isfile(os.path.join(cookbook, "metadata.rb")):
        raise ValueError("Cookbooks '%s' не містить файл
        metadata.rb" % cookbook)
knife cookbooks upload % cookbook
```



Виконання конфігурації на вузлах (nodes)

- ☐ Виконання конфігурації на вузлах (nodes) - це процес застосування Cookbooks та інших файлів конфігурації до систем.
- ☐ Виконання конфігурації на вузлах (nodes) виконується за допомогою інструментів Chef, таких як Chef Client або Chef Server.
- ☐ Виконання конфігурації на вузлах (nodes) включає в себе такі етапи:
 - Запуск Chef Client на вузлах (nodes).
 - Застосування Cookbooks та інших файлів конфігурації до систем.
- ☐ Запуск Chef Client на вузлі (node)
 - `chef-client`



Версіонування Cookbooks

- ❑ Версіонування Cookbooks – це процес надання Cookbooks унікальних версій.
- ❑ Версіонування Cookbooks допомагає відстежувати зміни в Cookbooks і гарантувати, що всі вузли (nodes) отримують однакові версії Cookbooks.
- ❑ Версія Cookbooks складається з трьох частин:
 - **Основна версія:** Основна версія вказує на основні зміни в Cookbooks.
 - **Підверсія:** Підверсія вказує на менші зміни в Cookbooks.
 - **Номер випуску:** Номер випуску вказує на нову версію Cookbooks.
- ❑ Приклад
 - 1.0.0
 - Ця версія Cookbooks вказує на те, що Cookbooks є першою основною версією з підверсією 0 і номером випуску 0.



Тестування конфігурацій

- ☐ Тестування конфігурацій – це процес перевірки того, що конфігурації працюють належним чином.
- ☐ Тестування конфігурацій допомагає запобігти помилкам у конфігураціях і гарантувати, що системи налаштовані належним чином.



Інструменти для тестування

- ❑ Існує багато інструментів, які можна використовувати для тестування конфігурацій.
- ❑ Деякі популярні інструменти для тестування конфігурацій включають:
 - `ChefSpec` – це інструмент для тестування конфігурацій `Chef`. Він використовує `Ruby` для створення тестових сценаріїв, які перевіряють, що ресурси `Chef` налаштовані належним чином.
 - `ServerSpec` – це інструмент для тестування конфігурацій на серверах. Він використовує `Ruby` для створення тестових сценаріїв, які перевіряють, що системи відповідають певним специфікаціям.
 - `InSpec` – це інструмент для тестування конфігурацій на основі `Ruby`. Він використовує `Ruby` для створення тестових сценаріїв, які перевіряють, що системи відповідають певним вимогам.



Реальний приклад

```
# Метадані Cookbooks
```

```
name "django"
```

```
version "1.0.0"
```

```
maintainer "YD"
```

```
# Підготовка системи
```

```
run_list "apt", "python"
```

```
# Встановлення Python
```

```
apt_package "python" do
```

```
  action :install
```

```
end
```

```
# Встановлення Django
```

```
python_package "django" do
```

```
  action :install
```

```
end
```

```
# Створення каталогу для Django
```

```
directory "/srv/django" do
```

```
  owner "root"
```

```
  group "root"
```

```
  mode "0755"
```

```
  action :create
```

```
end
```

```
# Скопійовані файли Django
```

```
cookbook_file "/srv/django/settings.py" do
```

```
  source "settings.py"
```

```
  owner "root"
```

```
  group "root"
```

```
  mode "0644"
```

```
end
```

```
# Запуск сервера Django
```

```
service "unicorn" do
```

```
  action :start
```

```
end
```



Вигоди від використання Chef

- ❑ Chef - це потужний інструмент, який може принести багато переваг для організацій, які використовують його для управління інфраструктурою.
- ❑ Деякі з переваг від використання Chef включають:
 - **Покращена точність і надійність:** Chef може допомогти вам гарантувати, що ваші системи налаштовані належним чином і працюють безперебійно.
 - **Зменшення витрат:** Chef може допомогти вам зменшити витрати на управління інфраструктурою, автоматизуючи завдання та зменшуючи необхідність ручного втручання.
 - **Покращена масштабованість:** Chef може масштабуватися для задоволення потреб великих організацій.
 - **Покращена безпека:** Chef може допомогти вам підвищити безпеку ваших систем, застосовуючи стандартні політики безпеки.



Перспективи розвитку інфраструктури

- ❑ Інфраструктура майбутнього буде більш складною і диверсифікованою, ніж сьогодні.
- ❑ Chef може допомогти організаціям впоратися з цими викликами, надаючи гнучкі та масштабовані можливості управління інфраструктурою.
- ❑ Деякі конкретні перспективи для Chef включають:
 - **Широке застосування штучного інтелекту та машинного навчання в інфраструктурі:** Chef може бути використаний для автоматизації завдань, які в даний час виконуються людьми, а також для створення нових можливостей, таких як самовідновлювальна інфраструктура.
 - **Збільшення використання хмарних технологій:** Chef може допомогти організаціям управляти своїми хмарами, забезпечуючи уніфіковану платформу для всіх їхніх ресурсів.
 - **Збільшення використання контейнерів:** Chef може допомогти організаціям управляти своїми контейнерами, забезпечуючи гнучкість і масштабованість.



Висновок. Ключові моменти

- ❑ Chef – це потужний інструмент, який може принести багато переваг для організацій, які використовують його для управління інфраструктурою.
- ❑ Chef може допомогти організаціям покращити точність і надійність, зменшити витрати, підвищити масштабованість і безпеку.
- ❑ Chef може допомогти організаціям впоратися з викликами, які постають перед інфраструктурою майбутнього.

Донецький національний технічний університет



Тема 6. Системи управління конфігураціями.

Лекція 10. Система управління конфігураціями Puppet

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

402

- ☐ Що таке Purrpet?
- ☐ Основна термінологія
- ☐ Архітектура Purrpet
- ☐ Основні компоненти Purrpet
- ☐ Налаштування компонент Purrpet



Що таке Puppet?

- ❑ Puppet – це інструмент автоматизації управління конфігурацією, який використовує модель описаних ресурсів. Це означає, що Puppet описує стан бажаного конфігурування для пристрою за допомогою мову опису ресурсів (RDL).
- ❑ Puppet використовує цей опис для автоматичного створення, управління та розгортання конфігурацій на пристроях. Це дозволяє організаціям скоротити витрати, підвищити надійність і безпеку, а також полегшити управління.
- ❑ Puppet використовується в широкому спектрі сучасних інфраструктурних рішень, включаючи:
 - Хмарні середовища
 - Віртуальні середовища
 - Фізичні середовища





Роль Puppet у сучасних інфраструктурних рішеннях

- ❑ Puppet використовується для управління конфігураціями в широкому спектрі сучасних інфраструктурних рішень, включаючи хмарні середовища, віртуальні середовища та фізичні середовища.
- ❑ Puppet допомагає організаціям скоротити витрати, підвищити надійність і безпечність, а також полегшити управління.
Наприклад, Puppet можна використовувати для:
 - Автоматизації розгортання програмного забезпечення
 - Оновлення системного програмного забезпечення
 - Управління системними параметрами
 - Виконання завдань адміністрування
- ❑ Puppet є потужним інструментом, який може допомогти організаціям покращити свою інфраструктуру.

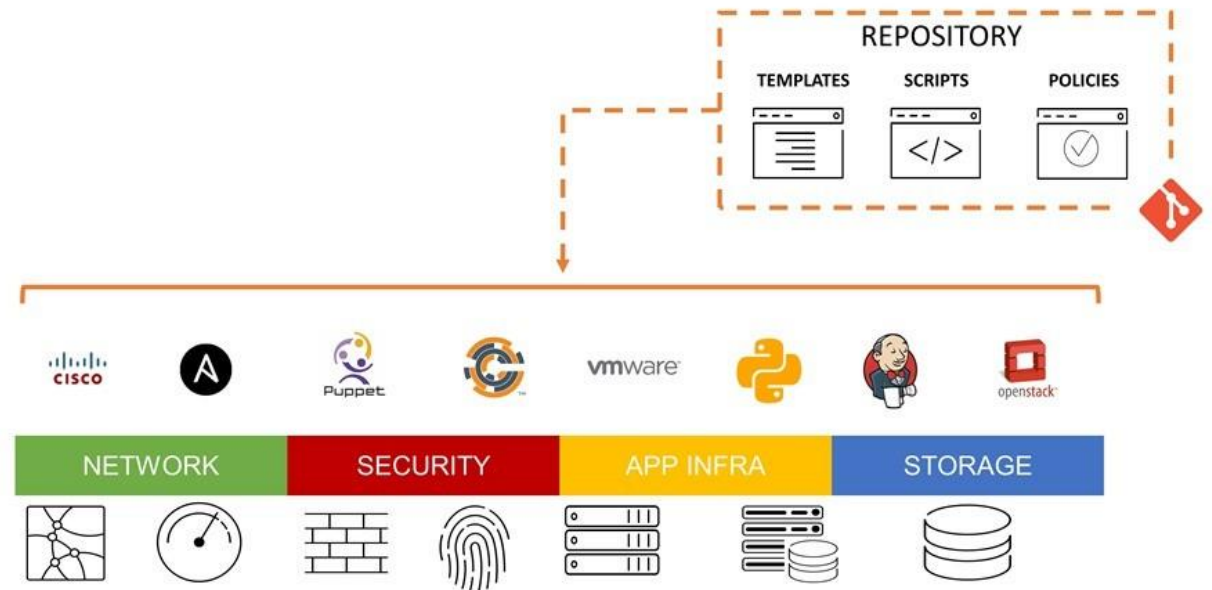


Інфраструктура як код

❑ Інфраструктура як код – це парадигма, яка розглядає інфраструктуру як програмний код. Це означає, що інфраструктуру можна описати за допомогою мови програмування.

❑ Інфраструктура як код дозволяє використовувати інструменти управління кодом, такі як Puppet, для автоматизації інфраструктури. Це дозволяє організаціям швидко і легко створювати, керувати та розгортати інфраструктуру.

INFRASTRUCTURE as CODE

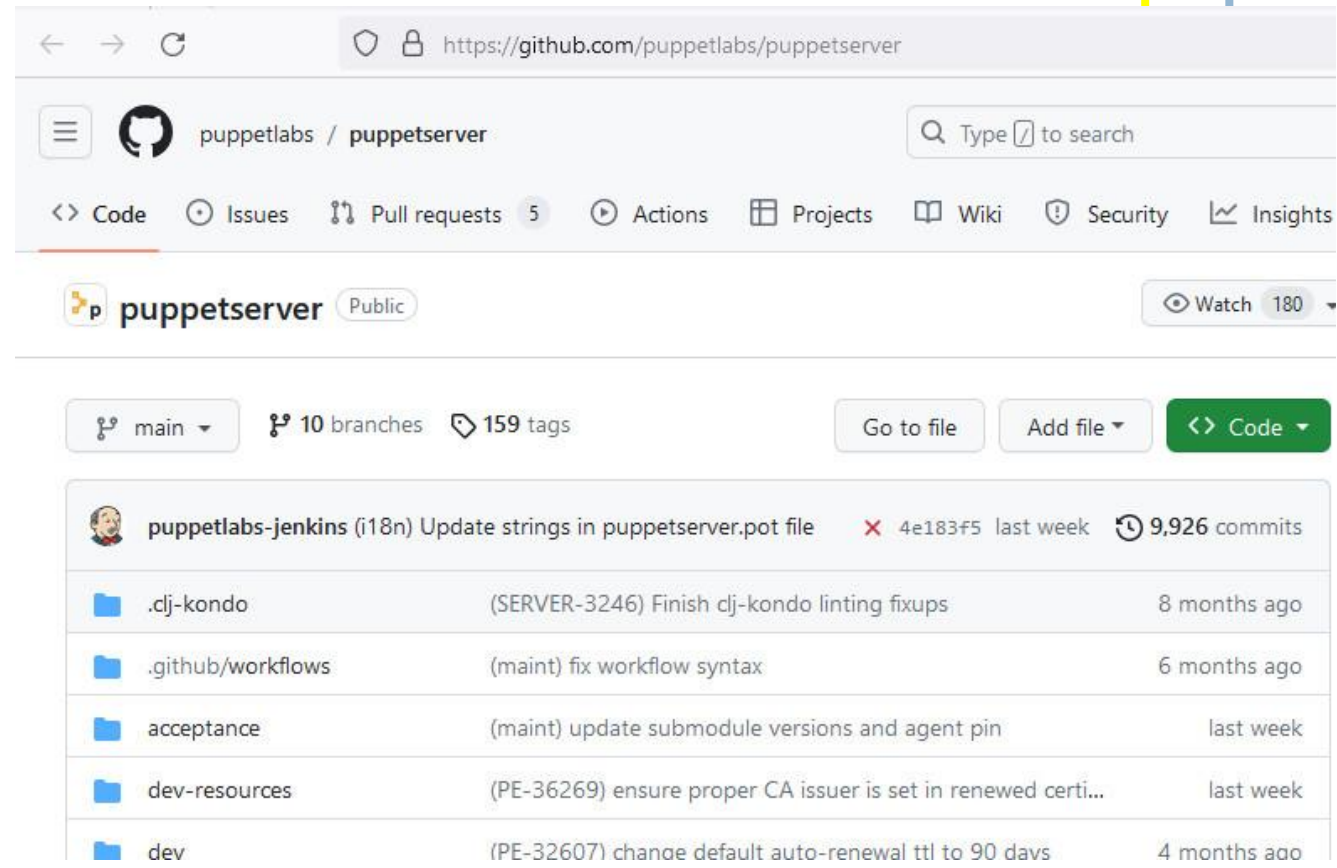




Декларативний підхід

❑ Декларативний підхід – це підхід до управління конфігурацією, який фокусується на тому, що потрібно зробити, а не на тому, як це зробити. Це означає, що Puppet описує бажаний стан конфігурації, а не те, як його досягти.

❑ Декларативний підхід дозволяє використовувати Puppet для створення конфігурацій, які є точними та послідовними. Це також полегшує управління конфігураціями, оскільки не потрібно турбуватися про те, як вони досягаються.



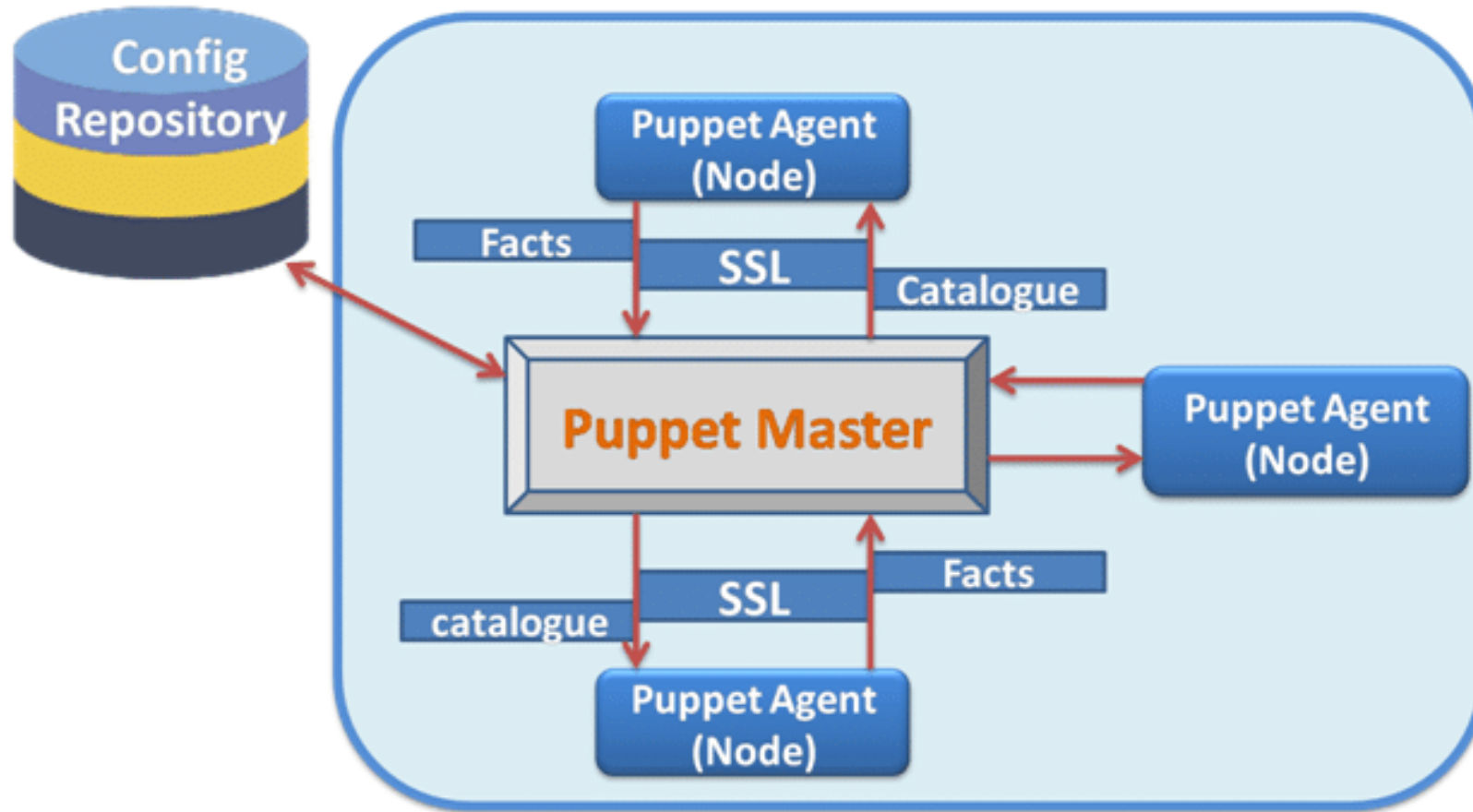


Автоматизація конфігурації

- ❑ Автоматизація конфігурації – це процес використання інструментів і методів для автоматичного створення, управління та розгортання конфігурацій.
- ❑ Puppet використовує модель описаних ресурсів для автоматизації конфігурації.



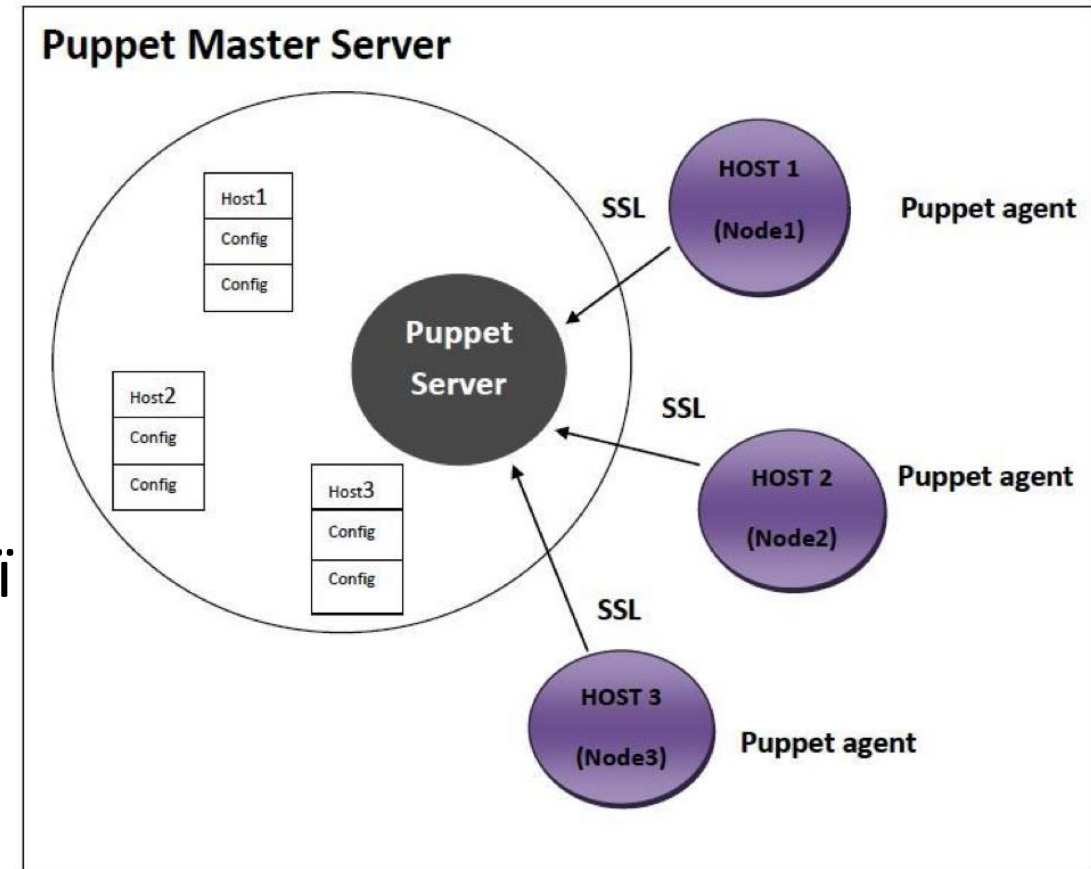
Архітектура Puppet





Puppet Master

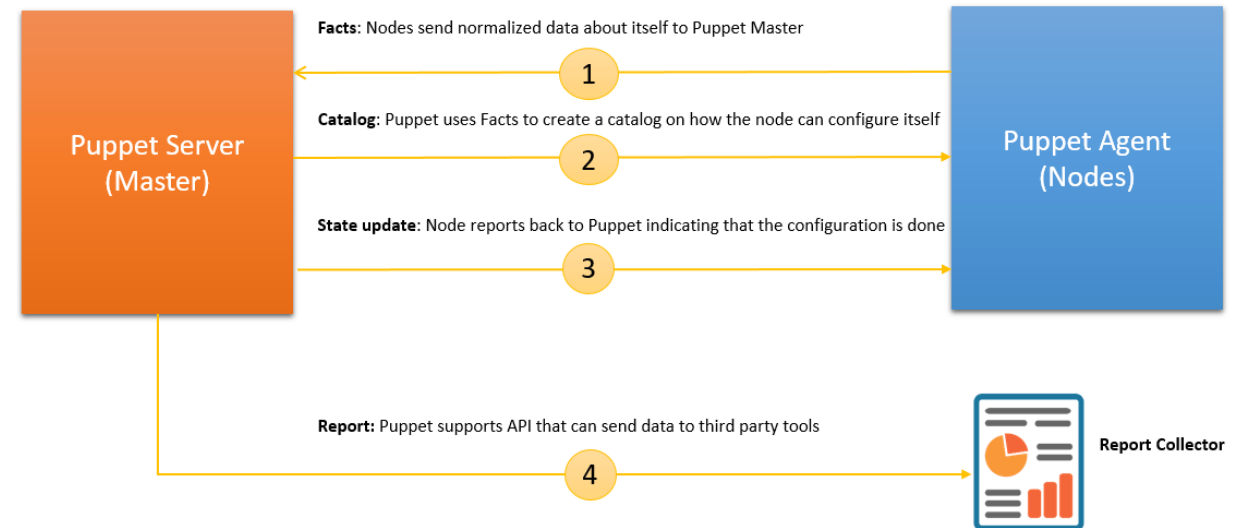
- ❑ Puppet Master – це центральний сервер, який керує Puppet Agent. Puppet Master зберігає файли Puppet, які описують конфігурацію пристроїв.
- ❑ Puppet Master використовує ці файли для визначення того, як повинні бути налаштовані пристрої. Він потім передає ці інструкції Puppet Agent, який виконує їх на пристроях.





Puppet Agent

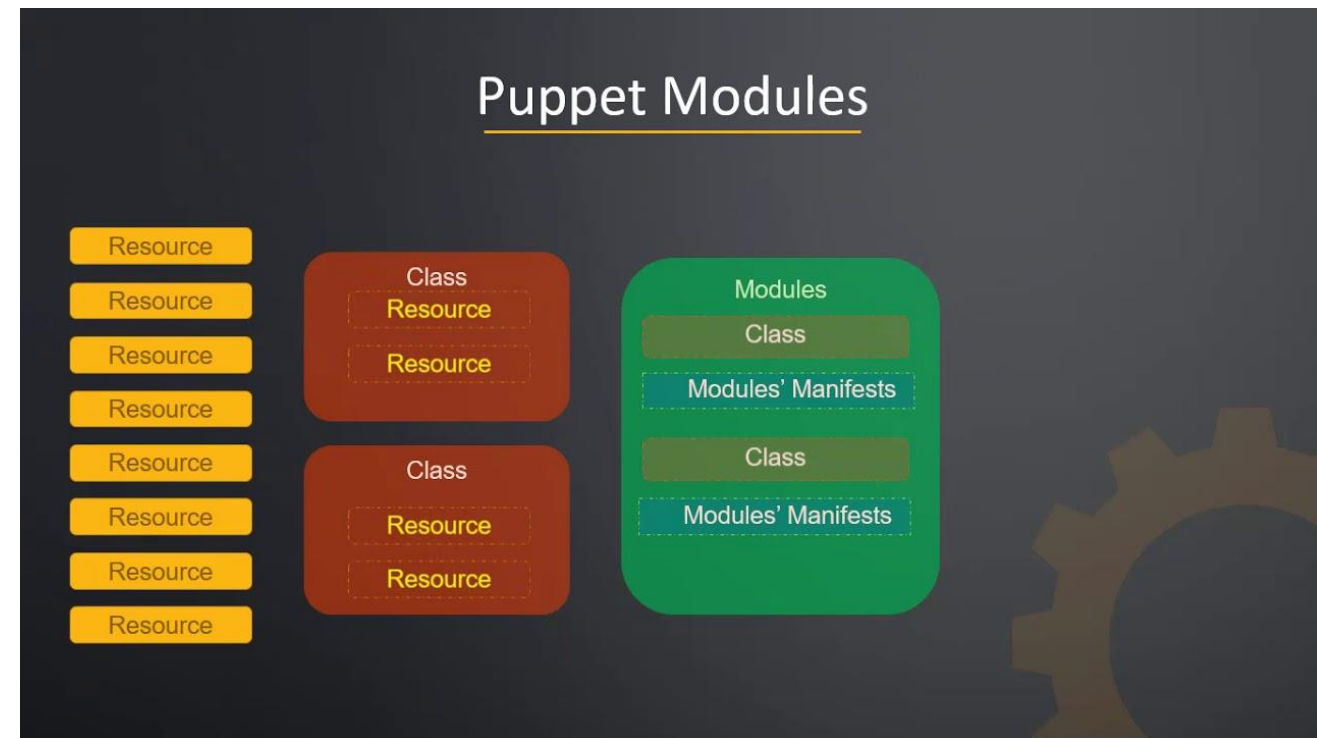
- ❑ Puppet Agent – це програмне забезпечення, яке працює на пристроях, які потрібно автоматизувати. Puppet Agent отримує інструкції від Puppet Master і виконує їх на пристроях.
- ❑ Puppet Agent використовується для виконання таких завдань, як:
 - Встановлення програмного забезпечення
 - Налаштування системних параметрів
 - Виконання завдань адміністрування





Класи та модулі Puppet

- ❑ Класи – це шаблони, які описують конфігурацію пристроїв. Вони використовуються для визначення того, як повинні бути налаштовані пристрої.
- ❑ Модулі – це групи класів, які використовуються для організації коду. Вони роблять код більш структурованим і легшим для читання та підтримки.





Ресурси Puppet

- ❑ Ресурси – це найнижчий рівень абстракції в Puppet. Вони описують окремі елементи конфігурації пристрою.
- ❑ Ресурси можуть бути використані для опису таких елементів, як:
 - Файли
 - Служби
 - Програмне забезпечення
 - Системні параметри



Факти (Facts) Puppet

- ❑ Факти – це дані про пристрої, які використовуються Puppet для визначення їхньої конфігурації. Факти збираються Puppet Agent.
- ❑ Puppet Agent використовує факти для визначення того, як повинні бути налаштовані пристрої. Наприклад, якщо факт показує, що на пристрої встановлено певне програмне забезпечення, Puppet Agent може використовувати цей факт для визначення того, чи потрібно оновлювати програмне забезпечення.



Мова опису конфігурації (Puppet DSL)

- ❑ Puppet DSL - це мова програмування, яка використовується для опису конфігурації пристроїв.
- ❑ Puppet DSL є декларативною мовою, тобто вона фокусується на тому, що потрібно зробити, а не на тому, як це зробити.
- ❑ Puppet DSL використовує такі ключові слова, як:
 - `class` - для визначення класу
 - `resource` - для визначення ресурсу
 - `require` - для вказівки, що один клас або ресурс залежить від іншого

```
1 # IIS Web Server
2 class web {
3   String $site_path = 'E:\\Dev\\Website\\',
4   String $website_name = 'new_website',
5   String $app_pool_name = 'app_pool',
6 }
7 windowsfeature[['Web-Server', 'Web-Asp-Net45']:
8   ensure => present
9 }
10
11 $name_regex = /^.+Website/
12
13 iis_application_pool { $app_pool_name:
14   ensure .....=> 'present',
15   state .....=> 'started',
16   managed_pipeline_mode .....=> 'Integrated',
17   managed_runtime_version .....=> 'v4.0',
18 }
19
20 iis_site { $website_name:
21   ensure .....=> 'started',
22   physicalpath .....=> $site_path,
```



Зборка каталогу

- ❑ Зборка каталогу – це процес, за допомогою якого Puppet збирає всі файли Puppet, які описують конфігурацію пристроїв. Зборка каталогу виконується Puppet Master.
- ❑ Puppet Master використовує файли Puppet для створення каталогу, який містить усі необхідні інструкції для Puppet Agent.



Визначення та виконання ресурсів

- ❑ Ресурси – це найнижчий рівень абстракції в Puppet. Вони описують окремі елементи конфігурації пристрою.
- ❑ Puppet Agent використовує ресурси для визначення та виконання дій на пристроях. Наприклад, Puppet Agent може використовувати ресурс для установки програмного забезпечення, налаштування системи або виконання завдання адміністрування.



Налаштування Puppet Master

- ❑ Налаштування Puppet Master включає в себе встановлення та конфігурацію сервера, а також створення каталогу для зберігання файлів Puppet.
- ❑ Для встановлення та налаштування Puppet Master слід виконати наступні команди:
 - `sudo yum install puppet`
 - створити файл `/etc/puppet/puppet.conf` і додати в нього наступний код:

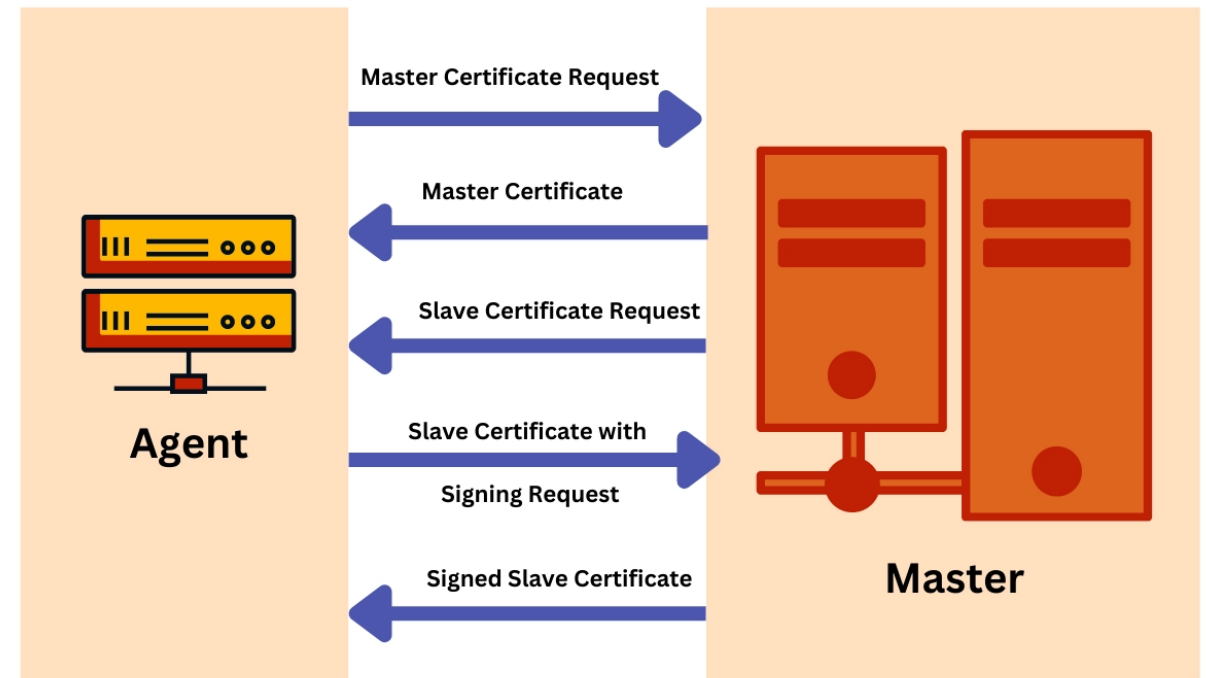
```
[main]
server = puppet
```
 - запустити Puppet Master:

```
sudo systemctl start puppetmaster
```



Реєстрація та звільнення Puppet Agent

- ❑ Реєстрація Puppet Agent – це процес, за допомогою якого Puppet Agent повідомляє Puppet Master про своє існування. Puppet Agent робить це, надсилаючи запит на Puppet Master.
- ❑ Puppet Master відповідає на запит Puppet Agent, надаючи йому інформацію про те, як він повинен бути налаштований.
- ❑ Звільнення Puppet Agent – це процес, за допомогою якого Puppet Agent повідомляє Puppet Master, що він більше не використовується. Puppet Agent робить це, надсилаючи запит на Puppet Master.
- ❑ Puppet Master відповідає на запит Puppet Agent, видаляючи Puppet Agent із свого списку пристроїв.





Основні команди Puppet

❑ Puppet має набір основних команд, які використовуються для управління Puppet Master та Puppet Agent. Ці команди можна використовувати для виконання таких завдань, як:

- Запуск Puppet: `puppet agent --verbose --onetime`
- Перегляд стану Puppet: `puppet agent --status`
- Відображення звітів: `puppet report`



Деякі інші основні команди Puppet

□ Деякі інші основні команди Puppet:

- `puppet master`: Керує Puppet Master.
- `puppet apply`: Виконує файли Puppet.
- `puppet module`: Керує модулями Puppet.
- `puppet parser`: Перетворює файли Puppet у формат, який можна використовувати для аналізу.



Визначення класів Puppet

❑ Класи визначаються в файлах Puppet з розширенням `.pp`.
Класи починаються з ключового слова `class`.

❑ Наприклад, наступний код визначає клас `my_server`:

```
class my_server {  
    # ...  
}
```



Використання модулів для організації коду

- ❑ Модулі - це спосіб організації коду в Puppet. Модулі можуть бути використані для повторення коду та організації коду.
- ❑ Модулі зазвичай зберігаються в каталогах. Кожен модуль має власний каталог, який містить файли Puppet, які описують класифікації, ресурси та інші елементи модуля.

```
# my_apache.pp
```

```
class my_apache {
```

```
    # Установить Apache
    package { 'apache2':
        ensure => 'present',
    }
```

```
    # Настроить Apache
    service { 'apache2':
        ensure => running,
        enable => true,
    }
```

```
}
```

```
# my_server.pp
```

```
include my_apache
```



Залежності між класами

❑ Класи можуть залежати від інших класів. Залежності дозволяють повторно використовувати код.

❑ Наприклад, ми можемо створити клас `my_server`, який залежить від класу `my_base`. Це означає, що клас `my_server` буде успадковувати все, що визначено в класі `my_base`.

```
# my_server.pp

class my_server {
  include my_base
}

class my_base {
  # ...
}
```

```
# my_web_server.pp

class my_web_server
{
  include my_server
  include my_apache
}

class my_server {
  # ...
}

class my_apache {
  # ...
}
```



Ресурси та їх атрибути

- ❑ Ресурси - це основні будівельні блоки Puppet. Ресурси описують, як Puppet повинен налаштувати пристрій.
- ❑ Ресурси мають атрибути, які описують їхню конфігурацію. Наприклад, ресурс `package` має атрибут `ensure`, який визначає, чи повинен пакет бути встановлений, видалений або оновлений.



Файлові ресурси

□ Файлові ресурси використовуються для створення, модифікації та видалення файлів.

□ Файлові ресурси мають такі атрибути:

- `path` - це шлях до файлу.
- `content` - це вміст файлу.
- `mode` - це права доступу до файлу.

□ Приклад ресурсу `file` для створення файлу:

```
file { '/etc/my_file':  
    content => 'Hello, world!',  
    mode => '0644',  
}
```



Ресурси служб

❑ Ресурси служб використовуються для керування службами.

❑ Ресурси служб мають такі атрибути:

- `ensure` - це стан служби.
- `enable` - це чи служба повинна бути включена.
- `start` - це чи служба повинна бути запущена.

❑ Приклад ресурсу `service` для запуску служби:

```
service { 'apache2':  
    ensure => running,  
    enable => true,  
}
```



Ресурси пакетів

❑ Ресурси пакетів використовуються для встановлення, видалення та оновлення пакетів.

❑ Ресурси пакетів мають такі атрибути:

- `ensure` - це стан пакету.
- `name` - це ім'я пакету.

❑ Приклад ресурсу `package` для встановлення пакету:

```
package { 'apache2':  
    ensure => present,  
}
```



Застосування конфігурацій до вузлів₁

- ❑ Застосування конфігурацій до вузлів – це процес, за допомогою якого Puppet забезпечує, щоб пристрої відповідали заданій конфігурації.
- ❑ Застосування конфігурацій до вузлів виконується за допомогою інструменту `puppet apply`. Інструмент `puppet apply` отримує файли Puppet, які описують конфігурацію пристроїв, і виконує дії, описані в цих файлах.



Застосування конфігурацій до вузлів₂

- ❑ Щоб застосувати цей клас до всіх наших серверів, ми можемо виконати наступну команду:

```
puppet apply my_apache.pp
```

- ❑ Якщо ми хочемо встановити Apache на конкретному вузлі, ми можемо використовувати команду `puppet apply` з параметром `node`:

```
puppet apply my_apache.pp --  
node my_server
```

- ❑ Ця команда застосує клас `my_apache` до вузла з ім'ям `my_server`.

```
class my_apache {  
  
    # Встановити Apache  
    package { 'apache2':  
        ensure => present,  
    }  
  
    # Налаштувати Apache  
    service { 'apache2':  
        ensure => running,  
        enable => true,  
    }  
}
```



Розгортання змін в інфраструктурі₁

- ❑ Розгортання змін в інфраструктурі – це процес, за допомогою якого Puppet вносить зміни в конфігурацію пристроїв.
- ❑ Розгортання змін в інфраструктурі може бути виконано за допомогою інструменту `puppet apply`. Для цього необхідно визначити зміни, які потрібно внести, у файлах Puppet.



Розгортання змін в інфраструктурі₂

- ❑ Припустимо, ми хочемо змінити порт, на якому працює веб-сервер Apache. Ми можемо зробити це, створивши зміну Puppet, яка описує зміну, яку ми хочемо внести.
- ❑ Щоб розгорнути цю зміну, ми можемо виконати наступну команду:
`puppet apply my_apache_port_change.pp`
- ❑ Ця команда розгорне зміну `my_apache_port_change` на всіх серверах, які зареєстровані у Puppet Master.
- ❑ Якщо ми хочемо розгорнути зміну на конкретному вузлі, ми можемо використовувати команду `puppet apply` з параметром `node`:
`puppet apply my_apache_port_change.pp --node my_server`
- ❑ Ця команда розгорне зміну `my_apache_port_change` на вузлі з ім'ям `my_server`.

```
class my_apache_port_change {  
  
    # Изменить порт Apache  
    file {  
        '/etc/apache2/ports.conf':  
            content => 'Listen 8080',  
        }  
    }  
}
```



Відміна змін

- ❑ Відміна змін – це процес, за допомогою якого Puppet повертає пристрої до попередньої конфігурації.
- ❑ Відміна змін може бути виконана за допомогою інструменту `puppet apply`. Для цього необхідно визначити зміни, які потрібно відмінити, у файлах Puppet.



Реальний приклад₁

```
class my_django {  
  
    # Установить Python  
    package { 'python3':  
        ensure => present,  
    }  
  
    # Установить Pip  
    package { 'python3-pip':  
        ensure => present,  
    }  
  
    # Установить Django  
    pip { 'django':  
        ensure => installed,  
    }  
  
    # Создать каталог для Django  
    file { '/var/www/my_django':  
        ensure => directory,  
    }  
  
    # Копировать Django-проект в каталог  
    file { '/var/www/my_django/my_project':  
        ensure => directory,  
        source => '/path/to/my_project',  
    }  
  
    # Установить службу Django  
    service { 'my_django':  
        ensure => running,  
        enable => true,  
    }  
}
```



Реальний приклад₂

□ Цей клас визначає дев'ять ресурсів:

- Ресурс `package` встановлює пакет `python3`.
- Ресурс `package` встановлює пакет `python3-pip`.
- Ресурс `pip` встановлює Django.
- Ресурс `file` створює каталог `/var/www/my_django`.
- Ресурс `file` копіює Django-проект в каталог `/var/www/my_django/my_project`.
- Ресурс `service` встановлює службу Django.



Реальний приклад₃

- ❑ Ще кілька можливих налаштувань, які можна використовувати для розгортання серверу Django:
- **Налаштування порту.** За замовчуванням сервер Django запускається на порту 8000. Ви можете змінити цей порт, використовуючи атрибут `port` ресурсу `service`.
 - **Налаштування середовища.** Ви можете налаштувати середовище Django, використовуючи атрибут `environment` ресурсу `file`. Наприклад, ви можете встановити змінну середовища `DJANGO_SETTINGS_MODULE` на значення `my_project.settings`.
 - **Налаштування безпеки.** Ви можете налаштувати безпеку Django, використовуючи атрибут `mode` ресурсу `file`. Наприклад, ви можете встановити права доступу до каталогу `/var/www/my_django` на 0755.

```
# Налаштувати змінну середовища
DJANGO_SETTINGS_MODULE
file { '/etc/default/my_django':
    ensure => file,
    content =>
"DJANGO_SETTINGS_MODULE=my_project.settings",
    owner => 'root',
    group => 'root',
    mode => '0644',
}
```



Вигоди від використання Puppet

- ❑ Puppet - це потужний інструмент для автоматизації управління конфігурацією. Puppet допомагає підвищити надійність і безпеку інфраструктури, а також скоротити витрати на адміністрування.
- ❑ Конкретні переваги Puppet:
 - **Підвищена надійність:** Puppet допомагає забезпечити, що всі пристрої в інфраструктурі будуть налаштовані однаково і точно. Це допомагає уникнути помилок і збоїв.
 - **Підвищена безпека:** Puppet може допомогти застосувати політики безпеки до всіх пристроїв в інфраструктурі. Це допомагає захистити інфраструктуру від атак.
 - **Скорочені витрати на адміністрування:** Puppet може допомогти автоматизувати завдання адміністрування, такі як установка програмного забезпечення, налаштування служб і управління мережею. Це звільняє адміністраторів для більш складних завдань.



Перспективи розвитку інфраструктури

- ❑ Puppet - це універсальний інструмент, який можна використовувати для автоматизації широкого спектру завдань. Puppet можна використовувати для підтримки інфраструктури в актуальному стані, а також для розгортання нових технологій.
- ❑ Перспективи розвитку інфраструктури з Puppet:
 - **Підтримка інфраструктури в актуальному стані:** Puppet можна використовувати для автоматичного оновлення програмного забезпечення, налаштування служб і управління мережею. Це допомагає підтримувати інфраструктуру в актуальному стані та запобігає виникненню проблем.
 - **Розгортання нових технологій:** Puppet можна використовувати для автоматизації процесу розгортання нових технологій. Це допомагає скоротити час і витрати, пов'язані з розгортанням нових технологій.

Донецький національний технічний університет



Тема 6. Системи управління конфігураціями.

Лекція 11. Система управління конфігураціями Ansible

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

439

- ☐ Вступ
- ☐ Архітектура Ansible
- ☐ Інвентаризація
- ☐ Playbook
- ☐ Шаблони в Ansible
- ☐ Модулі та директиви Ansible
- ☐ Ролі в Ansible
- ☐ Ansible Vault
- ☐ Інші можливості Ansible
- ☐ Ansible Tower
- ☐ Реальний приклад



Що таке Ansible?

- ❑ Ansible – це інструмент для автоматизації завдань з IT-інфраструктури. Він використовується для виконання завдань на віддалених системах без необхідності установки агентів на цих системах. Ansible працює за допомогою SSH і модулів Python.





Автоматизація з Ansible

❑ Ansible може використовуватися для автоматизації широкого спектру завдань, включаючи:

- налаштування систем
- розгортання програмного забезпечення
- управління конфігурацією
- моніторинг систем
- збирання даних

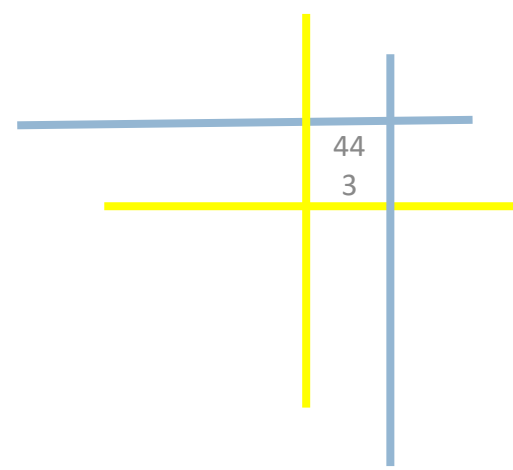
❑ Ansible є важливим інструментом для автоматизації інфраструктури, оскільки він може допомогти:

- зменшити витрати на ІТ-операції
- підвищити ефективність ІТ-команд
- покращити безпеку ІТ-систем



Як працює Ansible?

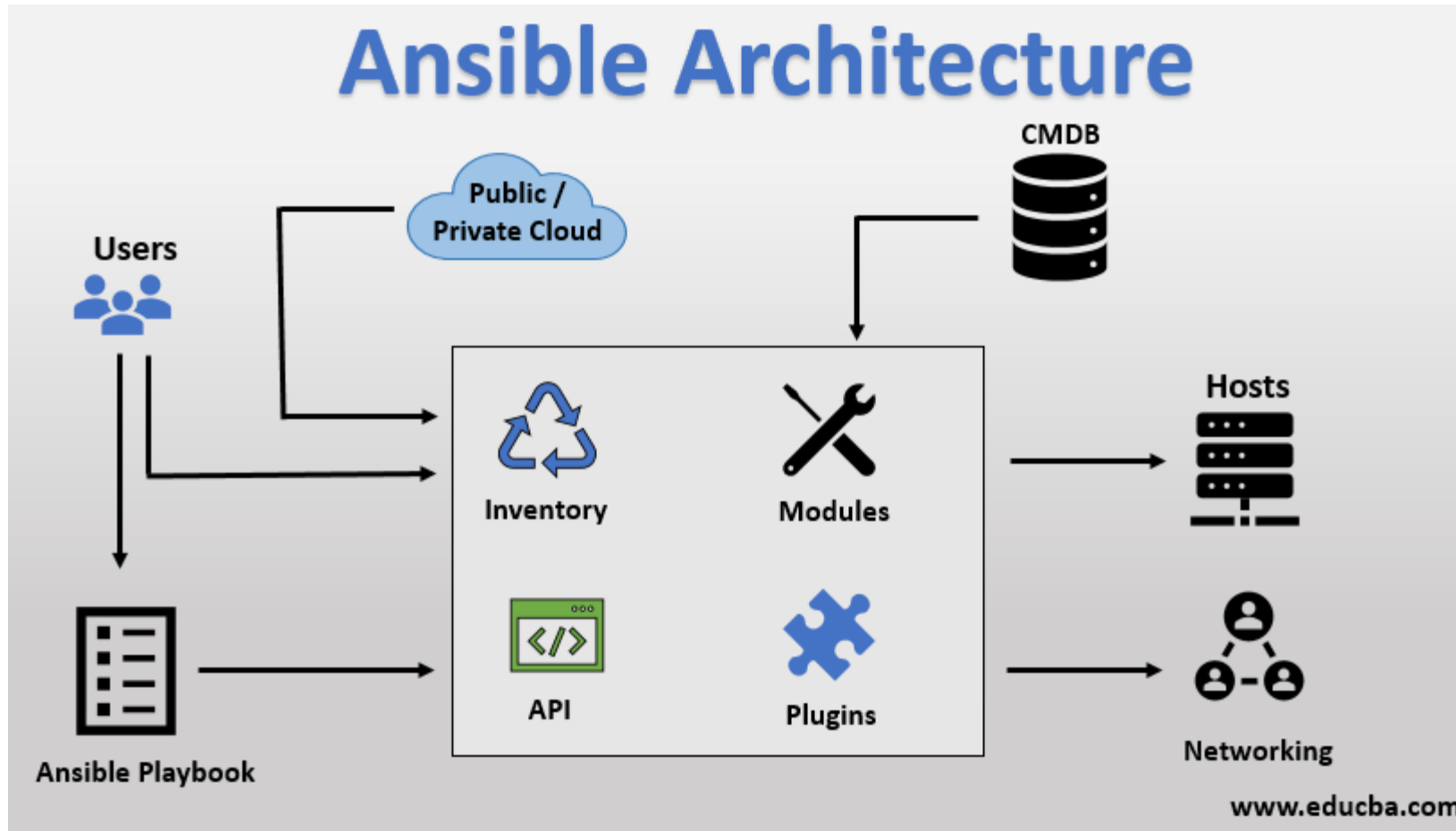
- ❑ Ansible працює за допомогою агент-less підходу. Це означає, що на управляємих вузлах не потрібно встановлювати жодних агентів. Ansible використовує SSH для підключення до управляємих вузлів і виконання завдань.
- ❑ Ansible заснований на моделях. Це означає, що Ansible працює з моделями стану системи, а не з конкретними командами. Це робить Ansible більш гнучким і універсальним, ніж інструменти автоматизації, які засновані на командах.
- ❑ Ansible використовує декларативний підхід. Це означає, що користувачі Ansible описують бажаний стан системи, а Ansible визначає, які дії потрібно виконати, щоб досягти цього стану. Це робить Ansible простішим у використанні, ніж інструменти автоматизації, які використовують імперативну модель.



Архітектура Ansible



Архітектура Ansible





Компоненти Ansible

- ❑ Ansible складається з трьох основних компонентів:
 - управляючий вузол
 - управляємий вузол
 - модулі
- ❑ Управляючий вузол є центральним компонентом архітектури Ansible. Він використовується для запуску завдань Ansible та управління ними.
- ❑ Управляємі вузли - це системи, на яких виконуються завдання Ansible. Вони можуть бути будь-якими системами, які підтримують SSH.
- ❑ Модулі - це Python-функції, які виконують завдання на управляємих вузлах. Вони надають доступ до широкого спектру операцій, включаючи запуск команд, завантаження файлів, зміну конфігурації та багато іншого.



Взаємодія компонентів Ansible

- ☐ Ansible працює шляхом обміну інформацією між управляючим вузлом і управляємими вузлами. Ця інформація включає в себе список завдань, які потрібно виконати, і дані, необхідні для виконання цих завдань.
- ☐ Управляючий вузол використовує SSH для підключення до управляємих вузлів. Після підключення він передає список завдань управляємим вузлам.
- ☐ Управляємі вузли використовують модулі Ansible для виконання завдань. Модулі отримують дані від управляючого вузла і використовують ці дані для виконання відповідних дій.
- ☐ Після завершення виконання завдань управляємі вузли повідомляють управляючий вузол про результати. Управляючий вузол зберігає ці результати для подальшого використання.
- ☐ Ця взаємодія дозволяє Ansible автоматизувати завдання на віддалених системах без необхідності установки агентів на цих системах.



Встановлення Ansible

- ❑ Linux: Ansible можна встановити за допомогою менеджера пакетів вашої дистрибутиву Linux. Наприклад, у дистрибутиві Ubuntu ви можете встановити Ansible за допомогою наступної команди:
 - `sudo apt install ansible`
- ❑ Windows: Ansible можна встановити за допомогою програми Chocolatey. Для встановлення Ansible за допомогою Chocolatey виконайте наступні команди:
 - `choco install ansible`
- ❑ macOS: Ansible можна встановити за допомогою менеджера пакетів Homebrew. Для встановлення Ansible за допомогою Homebrew виконайте наступну команду:
 - `brew install ansible`



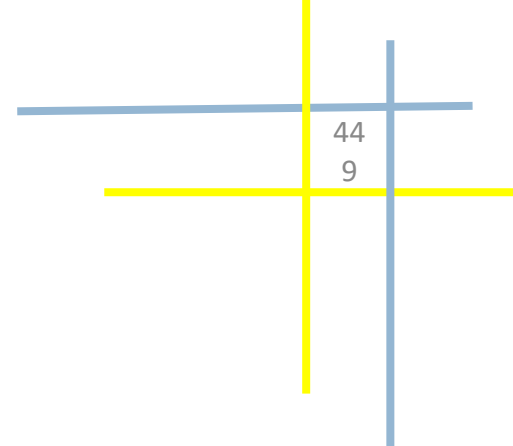
Перевірка версії та базові команди

❑ Щоб перевірити версію Ansible, виконайте наступну команду:

- `ansible --version`
- Ця команда виведе інформацію про версію Ansible, встановлену на вашому комп'ютері.

❑ Ansible має широкий спектр команд, які можна використовувати для автоматизації завдань. Ось кілька базових команд Ansible:

- `ansible all -m ping`: ця команда використовує модуль `ping` для перевірки доступності всіх управляємих вузлів
- `ansible all -m apt -a "name=nginx state=present"`: ця команда використовує модуль `apt` для встановлення пакету `nginx` на всіх управляємих вузлах
- `ansible all -m file -a "path=/etc/motd state=touch"`: ця команда використовує модуль `file` для створення файлу `/etc/motd` на всіх управляємих вузлах



Інвентаризація



Інвентаризація₁

- ❑ Інвентаризація - це файл, який містить список управляємих вузлів. Ansible використовує інвентарний файл для визначення, які вузли потрібно автоматизувати.
- ❑ Інвентаризаційні файли можна створювати в різних форматах, включаючи:
 - YAML
 - JSON
 - CSV
 - INI
- ❑ Найпоширенішим форматом інвентарного файлу є YAML.
- ❑ Приклад інвентарного файлу в YAML-форматі:

```
# Список хостів
hosts:
    # Група хостів web
    web:
        - 192.168.1.100
        - 192.168.1.101

    # Група хостів db
    db:
        - 192.168.1.102
```



Інвентаризація₂

- ❑ Ansible також підтримує додаткові параметри та можливості визначення хостів. Деякі з них включають:
- Задання імені хосту: можна задати ім'я хосту для кожного хоста в інвентаризаційному файлі. Ім'я хоста можна використовувати для полегшення читання інвентарного файлу.

```
hosts:
```

```
# Хост з іменем web1
```

```
web1:
```

```
    ansible_host: 192.168.1.100
```

```
# Хост з іменем db1
```

```
db1:
```

```
    ansible_host: 192.168.1.102
```



Інвентаризація₃

❑ Задання груп хостів за іменами хоста: можна задати групи хостів за іменами хоста. Це може бути корисно, якщо необхідно групувати хости на основі їхньої функції або ролі.

```
hosts:
```

```
# Група хостів web з хостами  
web1 і web2
```

```
web:
```

- web1
- web2

```
# Група хостів db з хостом db1
```

```
db:
```

- db1



Інвентаризація₄

❑ Задання змінних для хостів: можна задати змінні для кожного хоста в інвентаризаційному файлі. Змінні можна використовувати для зберігання інформації про хости, наприклад, про їхні операційні системи, конфігурацію або інші параметри.

```
hosts:
```

```
# Хост з іменем web1 і змінною  
operating_system=centos7
```

```
web1:
```

```
ansible_host: 192.168.1.100  
operating_system: centos7
```

```
# Хост з іменем db1 і змінною  
database_name=mydb
```

```
db1:
```

```
ansible_host: 192.168.1.102  
database_name: mydb
```



Ad-Hoc команди₁

- ☐ Ad-Hoc команди - це одноразові команди, які можна використовувати для взаємодії з хостами. Ad-Hoc команди можна використовувати для виконання таких завдань, як:
- ☐ перевірка доступності хоста
- ☐ виконання команд на хостах
- ☐ зміна конфігурації хоста
- ☐ передача файлів на хости



Ad-Hoc команди₂

- ❑ Ad-Hoc команди можна запускати з командного рядка або з інтерактивного режиму Ansible.
- ❑ Щоб запустити Ad-Hoc команду з командного рядка, виконайте наступну команду:
 - `ansible [інтервал] -m [модуль] -a [параметри]`
 - інтервал - це список хостів, на яких потрібно виконати команду. Якщо інтервал не вказано, команда виконається на всіх хостах, які визначені в інвентарному файлі
 - модуль – це назва модуля, який потрібно використовувати для виконання команди
 - параметри – це параметри, які потрібно передати модулю



Ad-Hoc команди₃

□ Приклад запуску Ad-Hoc команди для перевірки доступності хоста:

- `ansible all -m ping`
- Ця команда виконає модуль `ping` на всіх хостах, які визначені в інвентарному файлі.

□ Приклад запуску Ad-Hoc команди для встановлення пакету `nginx` на всіх хостах:

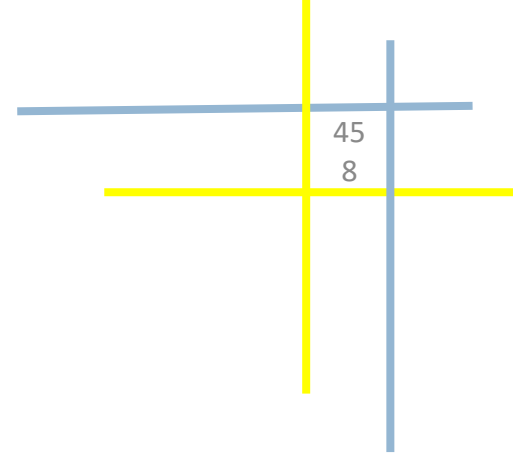
- `ansible all -m apt -a "name=nginx state=present"`
- Ця команда виконає модуль `apt` на всіх хостах, які визначені в інвентарному файлі. Модуль `apt` встановить пакет `nginx` на всіх хостах.



Ad-Hoc команди₄

❑ Ansible має широкий спектр модулів, які можна використовувати для виконання Ad-Hoc команд. Ось деякі з базових модулів:

- `ping`: цей модуль використовується для перевірки доступності хоста
- `apt`: цей модуль використовується для управління пакетами
- `file`: цей модуль використовується для управління файлами
- `user`: цей модуль використовується для управління користувачами
- `group`: цей модуль використовується для управління групами



Playbook



Playbook₁

❑ Playbook - це файл, який містить послідовність завдань, які потрібно виконати на управляємих вузлах. Playbookи можна використовувати для автоматизації складних завдань, які складно виконати за допомогою Ad-Hoc команд.

❑ Синтаксис playbook'ів

- Playbook'и написані в YAML-форматі. Синтаксис playbook'ів простий і зрозумілий
- Основні елементи playbook'а:
 - Ім'я: використовується для його ідентифікації
 - Інтервал: визначає, на яких хостів потрібно виконати playbook
 - Завдання: дії, які потрібно виконати на управляємих вузлах



Playbook₂

- ❑ Приклад простого playbook для автоматизації завдання встановлення пакету `nginx` на всіх хостах з групи `web`
- ❑ Щоб запустити playbook, виконайте наступну команду:

- `ansible-playbook playbook.yml`

```
---  
# Ім'я playbook'a  
- name: Install nginx  
# Інтервал  
hosts: web  
# Завдання  
tasks:  
  - name: Install nginx  
    apt:  
      name: nginx  
      state: present
```




Використання циклів у Playbook'ах

- ❑ Ansible має кілька циклів, які можна використовувати для автоматизації завдань, які повторюються. Ось деякі з найпопулярніших циклів:
 - `for`: цей цикл використовується для повторення завдання для кожного елемента в списку.
 - `while`: цей цикл використовується для повторення завдання, поки певна умова виконується.
 - `until`: цей цикл використовується для повторення завдання, поки певна умова не виконується.
- ❑ Цикли можна використовувати для автоматизації різних завдань, які повторюються.



Приклади використання циклів₁

❑ Оновлення всіх пакетів на всіх управляємих вузлах:

```
---
- hosts: all
  tasks:
    - name: Update all packages
      apt:
        update_cache: yes
        upgrade: yes

    - name: Update all packages for each operating system
      for os in "{{ ansible_os_family }}":
        apt:
          update_cache: yes
          upgrade: yes
          os: "{{ os }}"
```

- Цей playbook оновлює всі пакети на всіх управляємих вузлах. Він також використовує цикл `for` для оновлення пакетів для кожного типу операційної системи.



Приклади використання циклів₂

❑ Створення нових користувачів на всіх управляємих вузлах:

```
---
- hosts: all
  tasks:
    - name: Create new users
      user:
        name: "{{ item }}"
        state: present
      with_items:
        - user1
        - user2
        - user3
```

- Цей playbook створює нових користувачів на всіх управляємих вузлах. Він використовує цикл `with_items` для перерахування імен користувачів, які потрібно створити.



Тестування Playbook

- ❑ Тестування playbook'ів - це важливий процес, який допомагає гарантувати, що playbook'и працюють правильно та не завдадуть шкоди управляємим вузлам.
- ❑ Види тестування playbook'ів:
 - синтаксичне тестування
 - логічне тестування
 - інтеграційне тестування
 - системне тестування



Тестові фреймворки

❑ Для проведення тестування `playbook`'ів можна використовувати тестові фреймворки. Тестові фреймворки допомагають автоматизувати процес тестування та полегшують його виконання.

❑ Приклади тестових фреймворків:

- `Beaker`: цей тестовий фреймворк дозволяє тестувати `playbook`'и на віртуальних машинах.
- `Testinfra`: цей тестовий фреймворк дозволяє тестувати `playbook`'и на реальних управляємих вузлах.
- `Ansible-lint`: цей інструмент дозволяє перевіряти синтаксис `playbook`'ів.



Відлагодження Playbook

❑ Ansible має кілька інструментів, які можна використовувати для відлагодження playbook'ів. Ось кілька параметрів, які можна використовувати для відлагодження Ansible:

- `--check`: цей параметр дозволяє Ansible перевірити playbook'и на наявність помилок, не виконуючи їх фактично.
- `--limit`: цей параметр дозволяє Ansible виконувати playbook'и тільки на певних вузлах.
- `--debug`: цей параметр дозволяє Ansible виводити додаткову інформацію про виконання playbook'ів.



Параметр `--check`

❑ Параметр `--check` дозволяє Ansible перевірити playbook'и на наявність помилок, не виконуючи їх фактично. Це може бути корисно для виявлення синтаксичних помилок або помилок у логіці playbook'a.

- `ansible-playbook playbook.yml --check`



Параметр `--limit`

❑ Параметр `--limit` дозволяє Ansible виконувати playbook'и тільки на певних вузлах. Це може бути корисно для відлагодження playbook'а на конкретному вузлі або для відключення певних вузлів від виконання playbook'а.

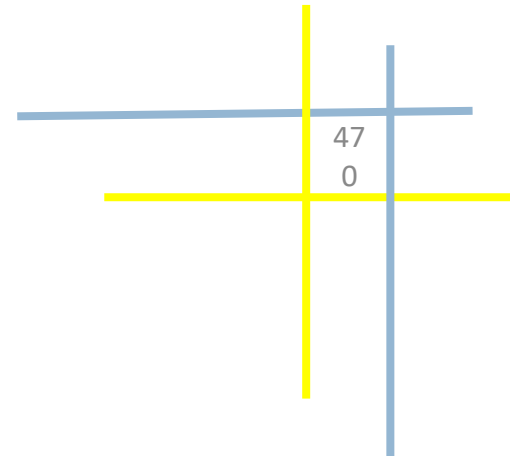
- `ansible-playbook playbook.yml --limit webserver`



Параметр `--debug`

❑ Параметр `--debug` дозволяє Ansible виводити додаткову інформацію про виконання playbook'a. Це може бути корисно для відстеження того, що відбувається під час виконання playbook'a.

- `ansible-playbook playbook.yml --debug`



Шаблони Ansible



Шаблони в Ansible

- ❑ Шаблони в Ansible - це текстові файли, які можна використовувати для створення конфігураційних файлів, веб-сторінок та інших документів. Ansible може використовувати шаблони для генерування конфігураційних файлів на керованих вузлах.
- ❑ Ansible підтримує два типи шаблонів:
 - Шаблони `Jinja2`: цей тип шаблонів є найбільш популярним і використовується для створення конфігураційних файлів, веб-сторінок та інших документів.
 - Шаблони `Mustache`: цей тип шаблонів є менш популярним, але він може бути корисний для створення простих шаблонів.



Використання шаблонів для створення конфігураційних файлів₁

❑ Ansible можна використовувати для генерування конфігураційних файлів на керованих вузлах. Для цього потрібно використовувати шаблон `Jinja2` і вказати Ansible, який модуль потрібно використовувати для створення конфігураційного файлу.

❑ Приклад використання шаблону `Jinja2` для генерування конфігураційного файлу Apache:

```
---
- hosts: all
  tasks:
    - name: Generate Apache configuration file
      template:
        src: apache.j2
        dest: /etc/apache2/apache2.conf
        owner: root
        group: root
        mode: 0644

  vars:
    server_name: www.example.com
    document_root: /var/www/html
```



Використання шаблонів для створення конфігураційних файлів₂

- ❑ Представлений playbook генерує конфігураційний файл Apache на всіх керованих вузлах. Шаблон `Apache.j2` містить наступний код:

```
<VirtualHost *:80>
    ServerName {{ server_name }}
    DocumentRoot {{ document_root }}
</VirtualHost>
```



Використання шаблонів для створення конфігураційних файлів₃

❑ Ansible можна використовувати для генерування веб-сторінок. Для цього потрібно використовувати шаблон Jinja2 і вказати Ansible, який веб-сервер потрібно використовувати для розгортання веб-сторінки.

❑ Приклад використання шаблону Jinja2 для генерування веб-сторінки:

```
---
- hosts: all
  tasks:
    - name: Generate web page
      template:
        src: index.j2
        dest: /var/www/html/index.html
        owner: root
        group: root
        mode: 0644

vars:
  title: My Website
  content: This is my website.
```



Використання шаблонів для створення конфігураційних файлів₄

- ❑ Представлений playbook генерує веб-сторінку на всіх керованих вузлах. Шаблон `index.j2` містить наступний код:

```
<!DOCTYPE html>
<html>
<head>
    <title>{{ title }}</title>
</head>
<body>
    {{ content }}
</body>
</html>
```



Загальні рекомендації з використання шаблонів в Ansible

- ☐ Використовуйте шаблони `Jinja2`, якщо потрібні потужні можливості для створення конфігураційних файлів, веб-сторінок та інших документів.
- ☐ Використовуйте шаблони `Mustache`, якщо потрібні прості шаблони.
- ☐ Використовуйте змінні в шаблонах для зберігання інформації, яка може змінюватися в залежності від ситуації.
- ☐ Використовуйте фільтри в шаблонах для модифікації тексту в шаблоні.



Розширені можливості шаблонів

- ❑ Ansible шаблони мають ряд розширених можливостей, які можна використовувати для створення більш складних і потужних шаблонів. Ось кілька з цих можливостей:
- *Фільтри*: можна використовувати для модифікації тексту в шаблоні.
 - *Змінні*: можна використовувати для зберігання інформації, яка може змінюватися в залежності від ситуації.
 - *Умовні блоки*: можна використовувати для виконання різних дій залежно від умов.
 - *Цикли*: можна використовувати для повторення фрагментів шаблону.



Використання фільтрів у шаблонах

❑ Фільтри можна використовувати для модифікації тексту в шаблоні. Ось кілька прикладів використання фільтрів:

- Виведення тексту в верхньому регістрі:

```
{{ message | upper }}
```

- Виведення перших 10 символів тексту:

```
{{ message | first 10 }}
```

- Заміна всіх пропусків на символи табуляції:

```
{{ message | replace: " ", "\t" }}
```



Використання змінних у шаблонах

❑ Змінні можна використовувати для зберігання інформації, яка може змінюватися в залежності від ситуації. Ось кілька прикладів використання змінних:

- Виведення імені користувача:

```
{{ ansible_user }}
```

- Виведення поточної дати та часу:

```
{{ ansible_date_time.iso8601 }}
```

- Виведення результату виконання завдання:

```
{{ task_result.stdout }}
```



Умовні блоки у шаблонах

□ Умовні блоки можна використовувати для виконання різних дій залежно від умов. Ось кілька прикладів використання умовних блоків:

- Виведення повідомлення, якщо користувач є адміністратором:

```
{% if ansible_user == "root" %}  
    {{ message }}  
{% endif %}
```

- Виведення різних повідомлень для різних операційних систем:

```
{% if ansible_os_family == "RedHat" %}  
    {{ message_for_redhat }}  
{% else %}  
    {{ message_for_debian }}  
{% endif %}
```



Цикли у шаблонах

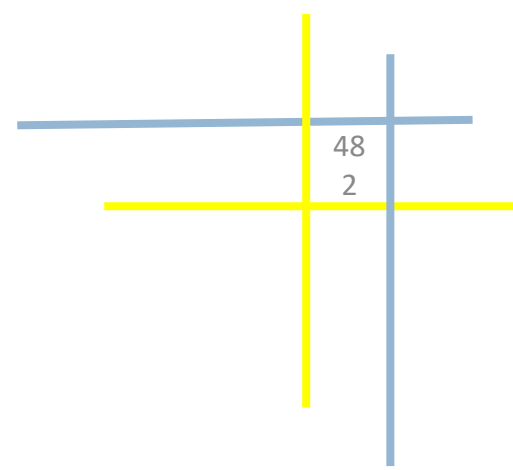
❑ Цикли можна використовувати для повторення фрагментів шаблону. Ось кілька прикладів використання циклів:

- Виведення списку імен користувачів:

```
{% for user in users %}  
    {{ user }}  
{% endfor %}
```

- Виведення таблиці з даними:

```
{% for row in data %}  
    | {{ row.column1 }} | {{ row.column2 }} |  
{% endfor %}
```



Модулі та директиви Ansible



Модулі Ansible

❑ Модулі - це Python-функції, які виконують завдання на управляємих вузлах. Ansible має широкий спектр модулів, які можна використовувати для виконання різних завдань, таких як:

- управління пакетами
- управління файлами
- управління користувачами
- управління групами
- управління конфігурацією

❑ Ось деякі з базових модулів Ansible:

- `ping`: цей модуль використовується для перевірки доступності хоста
- `apt`: цей модуль використовується для управління пакетами
- `file`: цей модуль використовується для управління файлами
- `user`: цей модуль використовується для управління користувачами
- `group`: цей модуль використовується для управління групами



Модулі для керування пакетами

- ❑ Ansible має кілька модулів для керування пакетами. Ось деякі з найпопулярніших модулів:
- `apt` : цей модуль використовується для управління пакетами в системах на базі Debian.
 - `yum` : цей модуль використовується для управління пакетами в системах на базі Red Hat.
 - `pacman` : цей модуль використовується для управління пакетами в системах на базі Arch Linux.



Модулі для керування сервісами

❑ Ansible має кілька модулів для керування сервісами. Ось деякі з найпопулярніших модулів:

- `service`: цей модуль використовується для управління сервісами в системах на базі Debian, Red Hat та Arch Linux.
- `systemd`: цей модуль використовується для управління сервісами в системах на базі `systemd`, таких як Red Hat, Fedora та CentOS.
- `upstart`: цей модуль використовується для управління сервісами в системах на базі `upstart`, таких як Ubuntu та Debian.



Приклади автоматизації оновлення програмного забезпечення₁

❑ Оновлення всіх пакетів на всіх управляємих вузлах:

- hosts: all

- tasks:

- name: Update all packages

- apt:

- update_cache: yes

- upgrade: yes



Приклади автоматизації оновлення програмного забезпечення₂

❑ Оновлення певного пакету на всіх управляємих вузлах:

```
- hosts: all
  tasks:
    - name: Update nginx
      apt:
        name: nginx
        update_cache: yes
        upgrade: yes
```



Приклади автоматизації оновлення програмного забезпечення₃

❑ Оновлення певного пакету на певних управляємих вузлах:

```
- hosts: webservers
  tasks:
    - name: Update nginx
      apt:
        name: nginx
        update_cache: yes
        upgrade: yes
```



Директиви Ansible

- ☐ Директиви - це ключові слова, які використовуються для управління виконанням playbook'ів. Директиви можна використовувати для:
- ☐ визначення змінних
- ☐ управління умовами виконання завдань
- ☐ управління послідовністю виконання завдань



Приклади директив Ansible

☐ Визначення змінної:

```
variable: value
```

☐ Управління умовами виконання завдань:

```
- name: Task 1
```

```
condition: "{{ variable == 'value' }}"
```

☐ Управління послідовністю виконання завдань:

```
- name: Task 1
```

```
  priority: 10
```

```
- name: Task 2
```

```
  priority: 20
```



Особливості обробки змінних

❑ Змінні в Ansible можуть оброблятися в різних контекстах. Контекст - це область видимості змінної.

❑ Основні контексти:

- Global: ці змінні доступні в усіх playbook'ах та ролях
- Play: ці змінні доступні в межах одного playbook'a
- Task: ці змінні доступні в межах одного завдання
- Module: ці змінні доступні в межах одного модуля

❑ Особливості обробки змінних у різних контекстах:

- Global: оголошуються в файлі `ansible.cfg` або в змінній `ansible_vars`
- Play: оголошуються в блоці `vars` playbook'a
- Task: оголошуються в блоці `vars` завдання
- Module: оголошуються в блоці `vars` модуля



Приклад використання змінних у різних контекстах

```
---
- hosts: all
  vars:
    my_variable: "Hello world!"

  tasks:
    - name: Print global variable
      debug:
        msg: "{{ my_variable }}"

    - name: Print play variable
      vars:
        my_variable: "Goodbye world!"
      debug:
        msg: "{{ my_variable }}"
```

```
- name: Print task variable

  tasks:
    - name: Set task variable
      vars:
        my_variable: "See you soon!"

    - name: Print task variable
      debug:
        msg: "{{ my_variable }}"
```




Застосування змін до конфігураційних файлів₁

- ❑ Ansible можна використовувати для зміни конфігураційних файлів на хостах. Для цього можна використовувати модуль file.
- ❑ Модуль file має наступні можливості:
 - створення файлів
 - редагування файлів
 - видалення файлів
 - перевірка наявності файлів



Застосування змін до конфігураційних файлів₂

❑ Створення файлу:

- name: Create file
file:
 path: /etc/myfile
 state: touch
- Це завдання створить файл /etc/myfile.

❑ Редагування файлу:

- name: Edit file
file:
 path: /etc/myfile
 state: present
 content: "This is my file"
- Це завдання редагує файл /etc/myfile, додаючи в нього текст "This is my file".



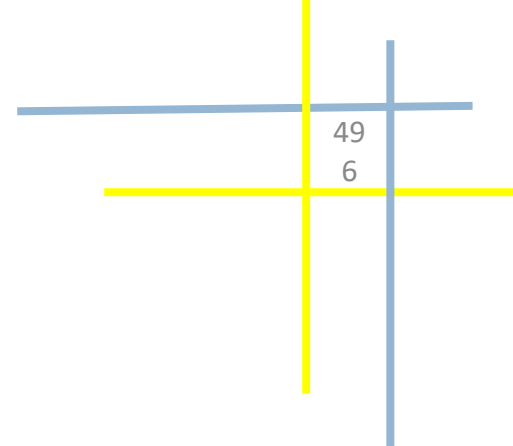
Застосування змін до конфігураційних файлів₃

❑Видалення файлу:

- name: Delete file
file:
 path: /etc/myfile
 state: absent
- Це завдання видаляє файл /etc/myfile.

❑Перевірка наявності файлу:

- name: Check if file exists
file:
 path: /etc/myfile
 state: directory
 register: result
- Це завдання перевіряє, чи існує файл /etc/myfile. Якщо файл існує, змінна `result` отримає значення `present`. В іншому випадку змінна `result` отримає значення `absent`.



Ролі в Ansible



Ролі в Ansible

- ❑ Roles - це спосіб логічно структурувати playbook'и та їхні компоненти. Roles можна використовувати для розширення playbook'ів, додаючи в них нові завдання, змінні та інші компоненти.
- ❑ Переваги використання ролей:
 - Roles допомагають зробити playbook'и більш структурованими та зрозумілими.
 - Roles дозволяють повторно використовувати код та завдання.
 - Roles допомагають зробити playbook'и більш масштабованими.



Структура ролей в Ansible

❑ Roles мають наступну структуру:

- `main.yml`: цей файл містить основну інформацію про роль, включаючи її ім'я, опис та інтервал
- `tasks`: цей каталог містить завдання, які виконуються в рамках ролі
- `handlers`: цей каталог містить хендлери, які запускаються в певних випадках, наприклад, після виконання завдання
- `vars`: цей каталог містить змінні, які використовуються в рамках ролі
- `templates`: цей каталог містить шаблони, які можна використовувати для створення файлів на управляємих вузлах

❑ Roles можна організувати в каталоги для більш ефективного управління. Наприклад, можна створити каталог для кожної групи хостів або для кожного типу завдання.



Приклад використання ролей

□ Приклад використання ролей:

- `hosts: web`

- `roles:`

- `webserver`

- Цей `playbook` виконає роль `webserver` на всіх хостах з групи `web`.



Використання External Roles₁

- ❑ Зовнішні ролі - це ролі, які зберігаються поза власним репозиторієм Ansible. Вони можуть зберігатися в інших репозиторіях Git, на веб-сайтах або в інших місцях.
- ❑ Переваги використання зовнішніх ролей:
 - полегшують розподіл і використання загальнодоступних рішень
 - зменшують обсяг роботи, який потрібно виконати для створення власних ролей
 - покращують повторюваність та надійність



Використання External Roles₂

- ❑ Щоб використовувати зовнішні ролі, потрібно спочатку їх зареєструвати в своєму Ansible. Це можна зробити, додавши посилання на репозиторій ролей у файл `roles_path` у файлі `ansible.cfg`.

```
roles_path:
```

- `/path/to/my/roles`
- `https://github.com/user/role.git`

- ❑ Після реєстрації ролі можна використовувати її в `playbook`'ах. Для цього просто додайте посилання на роль у список ролей, які потрібно виконати.

```
---
```

- `hosts: all`
- `roles:`
 - `my_role`



Приклад використання зовнішньої ролі

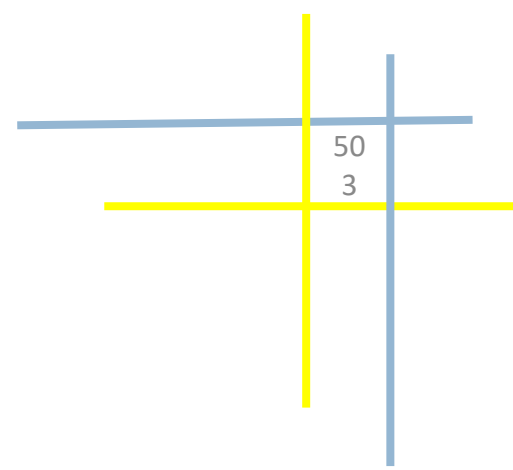
❑ Приклад використання зовнішньої ролі для установки та налаштування веб-сервера Apache:

- hosts: all

roles:

- role: ansible/apache

- Ця роль встановить Apache на всіх управляємих вузлах і налаштує його відповідно до стандартних налаштувань.



Ansible Vault



Ansible Vault

- ❑ Ansible Vault – це інструмент, який дозволяє безпечно зберігати конфіденційні дані, такі як паролі, секретні ключі та інші секрети.
- ❑ Ansible Vault має наступні команди:
 - `ansible-vault create`: ця команда створює новий зашифрований файл
 - `ansible-vault edit`: ця команда відкриває зашифрований файл для редагування
 - `ansible-vault encrypt`: ця команда шифрує файл
 - `ansible-vault decrypt`: ця команда розшифровує файл



Приклади використання Ansible Vault₁

❑ Шифрування пароля:

- `ansible-vault encrypt password.txt`
- Ця команда шифрує файл `password.txt`.

❑ Редагування зашифрованого файла:

- `ansible-vault edit password.txt`
- Ця команда відкриває файл `password.txt` для редагування.



Приклади використання Ansible Vault₂

❑ Використання зашифрованого файлу в playbook'i:

- hosts: all

- tasks:

- name: Get password

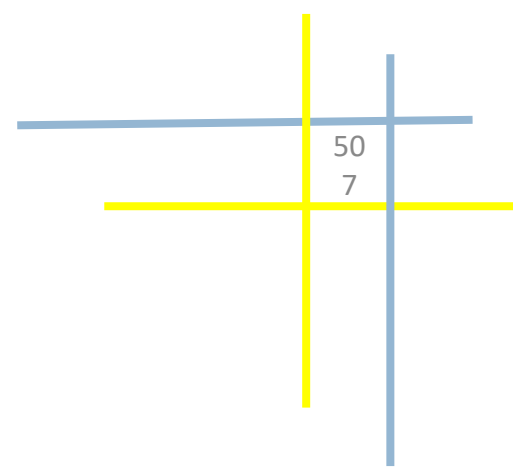
- vault_file: password.txt

- var: password

- name: Do something with password

- command: echo {{ password }}

- Цей playbook використовує зашифрований файл `password.txt` для отримання пароля. Потім пароль використовується для виконання команди `echo`.



Інші можливості Ansible



Динамічні інвентарні системи

- ❑ Динамічні інвентарні системи - це інструменти, які дозволяють автоматично збирати інформацію про управляємі вузли. Динамічні інвентарні системи можна використовувати для автоматизації завдання інвентаризації, а також для отримання актуальної інформації про управляємі вузли.
- ❑ Переваги використання динамічних інвентарних систем:
 - автоматичне збирання інформації про управляємі вузли
 - отримання актуальної інформації про управляємі вузли
 - збереження часу та зусиль
- ❑ Приклади динамічних інвентарних систем:
 - Ansible Tower
 - AWX
 - SaltStack
 - Puppet



Підключення до хмарних платформ

- ❑ Динамічні інвентарні системи можна використовувати для підключення до хмарних платформ. Це дозволяє автоматизувати завдання адміністрування хмарних ресурсів.
- ❑ Приклади підключення до хмарних платформ:
 - AWS
 - Azure
 - GCP
- ❑ Кілька прикладів того, як можна використовувати динамічні інвентарні системи:
 - *Автоматизація завдання інвентаризації.* Динамічна інвентарна система може автоматично збирати інформацію про управляємі вузли, наприклад, їхню IP-адресу, операційну систему та встановлені пакети. Ця інформація може бути використана для створення інвентарних файлів, які можна використовувати для виконання завдань Ansible.
 - *Отримання актуальної інформації про управляємі вузли.* Динамічна інвентарна система може регулярно збирати інформацію про управляємі вузли. Це дозволяє отримувати актуальну інформацію про вузли, навіть якщо вони були додані або видалені з мережі.
 - *Збереження часу та зусиль.* Динамічна інвентарна система може автоматизувати завдання інвентаризації, що може заощадити час та зусилля адміністраторів.



Оптимізація Ansible для великих інфраструктур

510

❑ Ось кілька рекомендацій щодо оптимізації Ansible для великих інфраструктур:

- використовуйте parallelism
- використовуйте асинхронність
- розподіліть навантаження
- використовуйте динамічні інвентарні системи
- використовуйте ролі



Інтеграція Ansible у процеси розробки програмного забезпечення

- ❑ Ansible можна використовувати для автоматизації різних завдань у життєвому циклі розробки програмного забезпечення. Наприклад, Ansible можна використовувати для:
- Автоматизації деплою програмного забезпечення
 - Автоматизації тестування програмного забезпечення
 - Автоматизації управління конфігурацією програмного забезпечення



Інтеграція Ansible у процес деплойменту₁

- ❑ Ansible можна використовувати для автоматизації процесу деплойменту програмного забезпечення. Для цього можна використовувати playbooks, які містять завдання для виконання різних етапів процесу деплойменту.



Інтеграція Ansible у процес деплоюменту₂

513

- hosts: all

tasks:

- name: Install required packages

apt:

name: "{{ item }}"

state: present

with_items:

- git

- python3-pip

- name: Clone repository

git:

repo: https://github.com/my-org/my-project.git

dest: /opt/my-project

- name: Install dependencies

pip:

requirements: /opt/my-project/requirements.txt

- name: Start application

service:

name: my-app

state: started



Інтеграція Ansible у процес тестування₁

- ❑ Ansible можна використовувати для автоматизації процесу тестування програмного забезпечення. Для цього можна використовувати playbook'и, які містять завдання для виконання різних етапів процесу тестування.



Інтеграція Ansible у процес тестування₂

- hosts: all

tasks:

- name: Install testing tools

apt:

name: "{{ item }}"

state: present

with_items:

- pytest

- coverage

- name: Run unit tests

pytest:

testpaths: /opt/my-project/tests

- name: Generate coverage report

coverage:

source: /opt/my-project

omit: /opt/my-project/tests



Інтеграція Ansible з іншими інструментами

- Ansible можна інтегрувати з іншими інструментами для автоматизації різних завдань. Ось кілька прикладів інтеграції Ansible з іншими інструментами:
- інтеграція Ansible з Jenkins
 - інтеграція Ansible з Git
 - інтеграція Ansible з іншими інструментами для управління конфігурацією



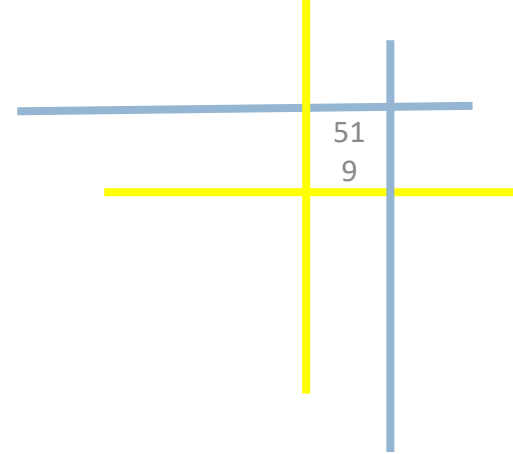
Взаємодія Ansible з контейнерами

- ❑ Ansible можна використовувати для взаємодії з контейнерами різними способами. Ansible може:
- Інсталювати та налаштовувати контейнерні платформи, такі як Docker і Kubernetes.
 - Створювати, запускати та зупиняти контейнери.
 - Виконувати завдання всередині контейнерів.
 - Перевіряти стан контейнерів.



Моніторинг та журналювання Ansible

- ❑ Ansible має вбудовані можливості моніторингу та журналювання, які можна використовувати для відстеження стану playbook'ів та керованих вузлів.
- ❑ Ansible зберігає журнали виконання playbook'ів у файлі `ansible.log`. Цей файл містить інформацію про всі виконані завдання, а також про будь-які помилки, які виникли.
- ❑ Ansible також може генерувати звіти про виконання playbook'ів. Ці звіти містять інформацію про час виконання playbook'ів, кількість виконаних завдань та будь-які помилки, які виникли.



Ansible Tower

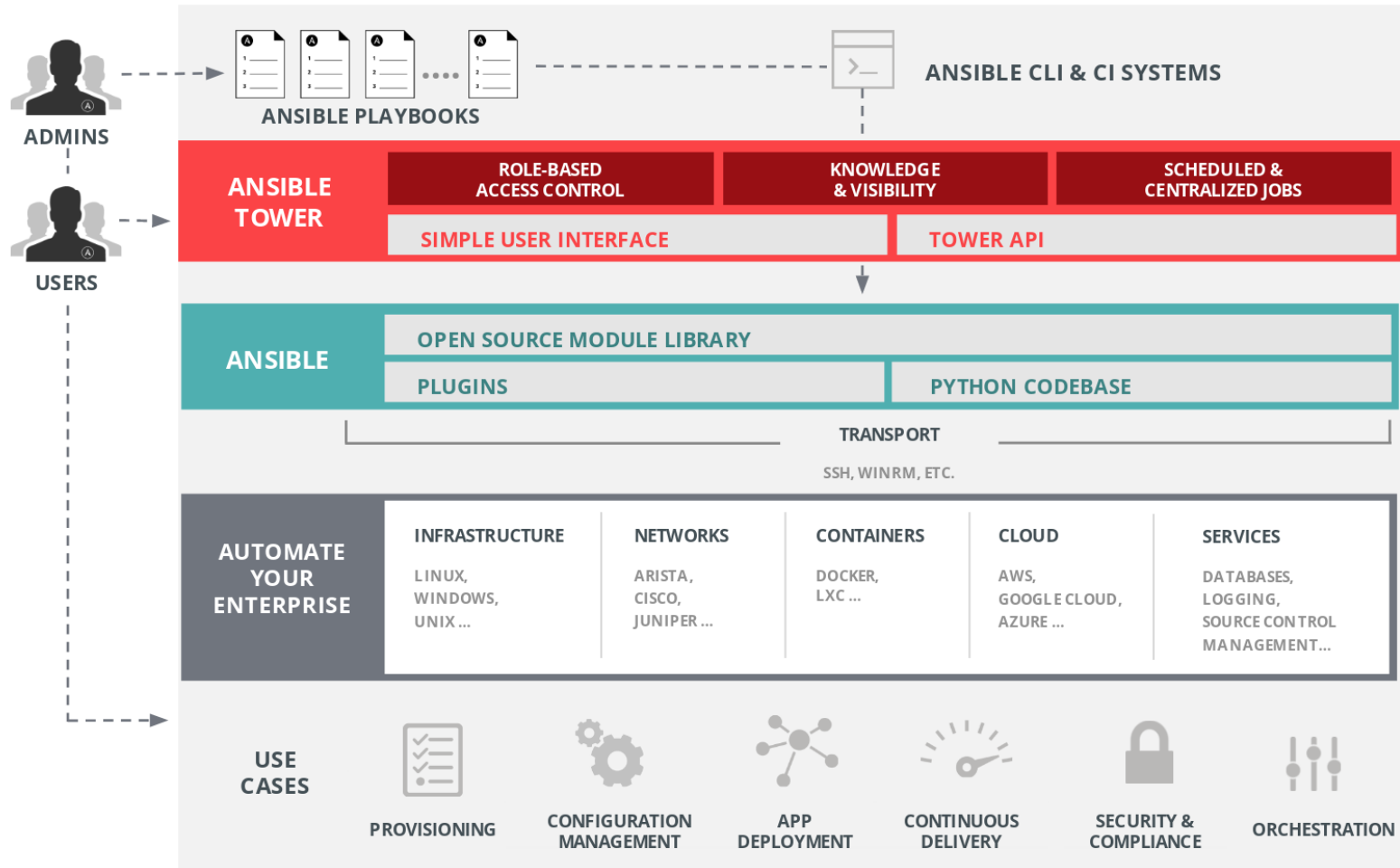


Ansible Tower

- ❑ Ansible Tower – це веб-інтерфейс для управління та моніторингу Ansible. Він дозволяє вам створювати, запускати та керувати playbook'ами Ansible, а також отримувати інформацію про стан ваших керованих вузлів.
- ❑ Графічний інтерфейс Ansible Tower має ряд переваг перед ручним керуванням Ansible:
 - *Зручність використання:* Ansible Tower значно спрощує створення та запуск playbook'ів Ansible.
 - *Ефективність:* Ansible Tower дозволяє автоматизувати завдання з керування та моніторингу Ansible, що може заощадити час та зусилля.
 - *Відстежуваність:* Ansible Tower дозволяє відстежувати стан керованих вузлів та історію виконаних playbook'ів.



Архітектура Ansible Tower





Основні функції Ansible Tower

❑ Ansible Tower має широкий спектр функцій, які можна використовувати для управління та моніторингу Ansible. Ось деякі з основних функцій:

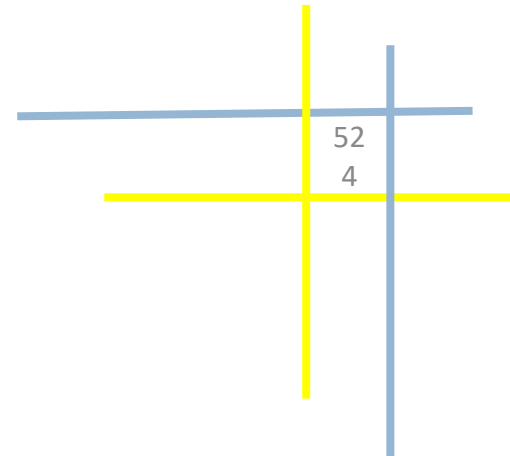
- *Створення та запуск playbook'ів*: Ansible Tower дозволяє створювати та запускати playbook'и Ansible за допомогою графічного інтерфейсу.
- *Керування активами*: Ansible Tower дозволяє керувати активами, такими як керовані вузли, групи керованих вузлів та змінні.
- *Менеджер завдань*: Ansible Tower дозволяє створювати та керувати завданнями, які представляють собою послідовність playbook'ів.
- *Менеджер звітів*: Ansible Tower дозволяє створювати звіти про стан ваших керованих вузлів та виконаних playbook'ів.



Інтерфейс Ansible Tower

The screenshot displays the Ansible Tower web interface. The top navigation bar includes the 'TOWER' logo, a user profile for 'admin', and icons for help, settings, and power. The left sidebar contains a menu with sections: VIEWS (Dashboard, Jobs, Schedules, My View), RESOURCES (Templates, Credentials, Projects, Inventories, Inventory Scripts), ACCESS (Organizations, Users, Teams), and ADMINISTRATION (Credential Types, Notifications, Management Jobs, Instance Groups, Applications, Settings). The main content area is titled 'JOBS' and shows a list of 13 jobs. The visible jobs are:

- 17 - Demo Job Template** (Playbook Run)
STARTED 9/11/2018 11:33:38 PM FINISHED 9/11/2018 11:33:43 PM
LAUNCHED BY admin
JOB TEMPLATE Demo Job Template
INVENTORY Demo Inventory
PROJECT Demo Project
CREDENTIALS Demo Credential
- 18 - Demo Project** (SCM Update)
STARTED 9/11/2018 11:33:33 PM FINISHED 9/11/2018 11:33:37 PM
PROJECT Demo Project
- 14 - Demo Job Template** (Playbook Run)
STARTED 9/11/2018 11:33:10 PM FINISHED 9/11/2018 11:33:16 PM
LAUNCHED BY admin
JOB TEMPLATE Demo Job Template
INVENTORY Demo Inventory
PROJECT Demo Project
CREDENTIALS Demo Credential
- 15 - Demo Project** (SCM Update)
STARTED 9/11/2018 11:33:05 PM FINISHED 9/11/2018 11:33:10 PM
PROJECT Demo Project
- 11 - Demo Job Template** (Playbook Run)



Реальний приклад використання Ansible



Реальний приклад₁

- ❑ Розглянемо приклад використання Ansible для розгортання інфраструктури на контейнерах.
- ❑ Нехай у нас є веб-сайт, який складається з наступних компонентів:
 - Веб-сервер: Apache або Nginx
 - База даних: MySQL або PostgreSQL
 - Прикладне програмне забезпечення: PHP, Python або Java
- ❑ Ми можемо використовувати Ansible для автоматизації розгортання цих компонентів на контейнерах.



Реальний приклад₂

❑ Ось покроковий план використання Ansible для розгортання інфраструктури на контейнерах:

- Інсталяція Ansible: спочатку нам потрібно встановити Ansible на нашому керуючому вузлі.
- Створення playbook'а: потім нам потрібно створити playbook'а, який буде виконувати завдання з розгортання контейнерів.
- Інсталювання Docker: на всіх керованих вузлах нам потрібно встановити Docker.
- Створення конфігурацій контейнерів: нам потрібно створити конфігурації контейнерів для кожного компонента інфраструктури.
- Розгортання контейнерів: запуск playbook'а Ansible розгорне контейнери на керованих вузлах.



Реальний приклад₃

```
- hosts: all
  become: yes
```

tasks:

```
- name: Install Docker
```

apt:

```
  name: docker.io
```

```
  state: present
```

```
- name: Create web server container
```

docker:

```
  image: nginx
```

```
  container_name: web-server
```

ports:

```
  - "80:80"
```

```
- name: Create database container
```

docker:

```
  image: mysql
```

```
  container_name: database
```

ports:

```
  - "3306:3306"
```

```
- name: Create application container
```

docker:

```
  image: php:7.4-apache
```

```
  container_name: application
```

ports:

```
  - "8080:80"
```



Реальний приклад₄

❑ Для кожного компонента інфраструктури нам потрібно створити конфігурацію контейнера. Конфігурація контейнера визначає, які програми будуть запущені в контейнері, які файли будуть копійовані в контейнер і які налаштування будуть застосовані до контейнера.

❑ Наприклад, ось конфігурація контейнера для веб-сервера:

```
version: "3.7"
services:
  web-server:
    image: nginx
    container_name: web-server
    ports:
      - "80:80"
    volumes:
      -
        ./nginx.conf:/etc/nginx/nginx.conf
```



Реальний приклад₅

❑ Для кожного компонента інфраструктури нам потрібно створити конфігурацію контейнера. Конфігурація контейнера визначає, які програми будуть запущені в контейнері, які файли будуть копійовані в контейнер і які налаштування будуть застосовані до контейнера.

❑ Наприклад, ось конфігурація контейнера для веб-сервера:

```
version: "3.7"
services:
  web-server:
    image: nginx
    container_name: web-server
    ports:
      - "80:80"
    volumes:
      -
        ./nginx.conf:/etc/nginx/nginx.conf
```



Реальний приклад₆

❑ Щоб розгорнути контейнери, слід виконати наступну команду:

- `ansible-playbook -i inventory playbook.yaml`
- Ця команда запустить playbook'a `playbook.yaml`, який розгорне контейнери на керованих вузлах.

Донецький національний технічний університет



Тема 7. Системи безперервної інтеграції.

Лекція 12. Система безперервної інтеграції Jenkins

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

532

- ☐ Вступ
- ☐ Архітектура Jenkins
- ☐ Конфігурація та робота зі сховищем коду
- ☐ Автоматична збірка та тестування
- ☐ Моніторинг та логування
- ☐ Використання контейнерів в Jenkins
- ☐ Масштабування Jenkins
- ☐ Інтеграція з засобами інфраструктури



Введення до Jenkins

- ☐ Jenkins - це безкоштовне та відкрите програмне забезпечення, яке використовується для автоматизації процесу розробки програмного забезпечення.
- ☐ Jenkins дозволяє розробникам автоматизувати такі завдання, як:
 - складання коду
 - тестування коду
 - розгортання коду
- ☐ Це дозволяє розробникам покращити свою продуктивність та якість коду. Наприклад, Jenkins може використовуватися для автоматизації процесу складання коду, що звільняє розробників від необхідності робити це вручну. Це може заощадити час розробників і зменшити ймовірність помилок.
- ☐ Jenkins також може використовуватися для автоматизації процесу тестування коду. Це може допомогти розробникам знайти та усунути помилки в коді на ранніх етапах розробки.
- ☐ Нарешті, Jenkins може використовуватися для автоматизації процесу розгортання коду. Це може допомогти розробникам швидко та легко розгорнути новий код в виробничу середовище.



Jenkins



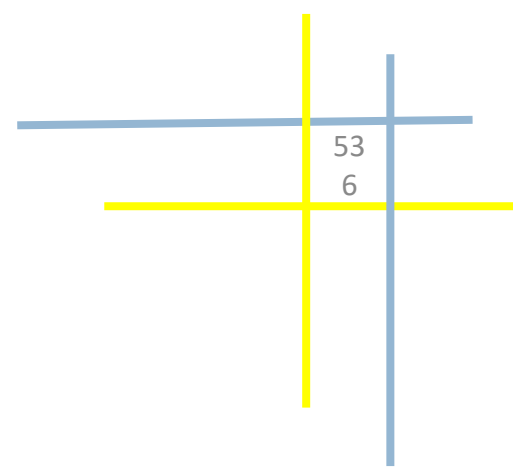
Історія Jenkins

- ❑ Jenkins був створений в 2004 році Косукє Кавагуті, розробником компанії Sun Microsystems. Jenkins був заснований на проекті Hudson, який був створений в 2005 році.
- ❑ У 2011 році Jenkins був переданий в фонд Apache Software Foundation. Це дозволило Jenkins стати більш відкритим і інклюзивним проектом.
- ❑ Jenkins швидко став одним з найпопулярніших інструментів автоматизації розробки. За даними Statista, у 2023 році Jenkins був використаний понад 25 мільйонами розробників.



Continuous Integration, Continuous Deployment, і їх важливість

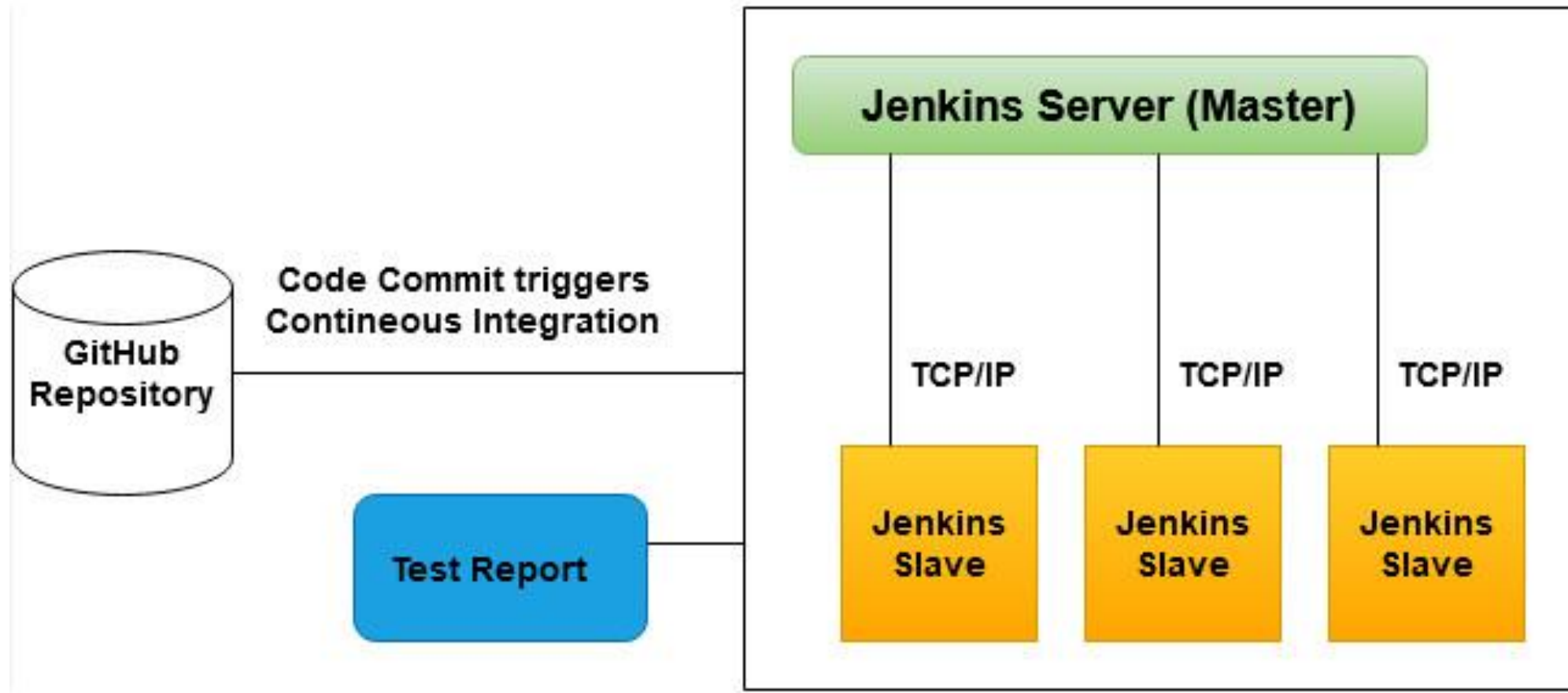
- ❑ Continuous Integration (CI) - це практика, яка полягає в тому, щоб часто і регулярно інтегрувати код розробників в єдиний репозиторій. Це допомагає виявляти та усувати помилки в коді на ранніх етапах розробки.
- ❑ Continuous Delivery (CD) - це практика, яка полягає в тому, щоб автоматично розгортати нові версії коду в виробничу середовище. Це допомагає забезпечити те, що код завжди готовий до випуску.



Архітектура Jenkins



Архітектура Jenkins₁





Архітектура Jenkins₂

❑ Jenkins – це розподілена система, яка складається з двох основних компонентів:

- Master: це центральний сервер, який керує всіма іншими компонентами Jenkins.
- Agents: це робочі машини, які виконують завдання, заплановані Master.

❑ Master:

- Відповідає за управління всіма іншими компонентами Jenkins.
- Зберігає всі конфігурації Jenkins, включаючи плани збірки, плагіни та інші налаштування.
- Запускає агентів та завдання.

❑ Agents:

- Виконують завдання, заплановані Master.
- Можуть бути фізичними або віртуальними машинами.
- Зазвичай працюють у віддаленому середовищі, наприклад, в хмарі.



Встановлення Jenkins

□ Процес встановлення:

- Завантажте Jenkins з веб-сайту Jenkins.
- Розпакуйте завантажений файл.
- Переконайтеся, що в системі встановлена JRE версії 8 або новіша.
- Запустіть файл `java -jar jenkins.war`.



Конфігурація Jenkins

❑ Основні параметри конфігурації:

- *Системні параметри*: ці параметри визначають основні налаштування Jenkins, такі як адреса, порт, пароль та інші.
- *Параметри безпеки*: ці параметри визначають правила безпеки для Jenkins, такі як політика паролів та інші.
- *Параметри плагінів*: ці параметри визначають налаштування плагінів, які встановлені в Jenkins.

❑ Налаштування базових параметрів:

- *Системні параметри*: ці параметри можна налаштувати на сторінці "System Settings".
- *Параметри безпеки*: ці параметри можна налаштувати на сторінці "Security".
- *Параметри плагінів*: ці параметри можна налаштувати на сторінці "Manage Plugins".



Інтеграція із засобами розробки

- ☐ *Системи управління джерелами:* Jenkins може бути інтегрований з різними системами управління джерелами, такими як Git, SVN та іншими.
- ☐ *Скриптові мови:* Jenkins може бути інтегрований з різними мовами сценаріїв, такими як Groovy, Shell та іншими.
- ☐ *Бібліотеки:* Jenkins може бути інтегрований з різними бібліотеками, такими як JUnit, Cucumber та іншими.



Інтеграція з іншими інструментами

- ❑ Jenkins можна інтегрувати з іншими інструментами розробки, щоб покращити співпрацю та автоматизацію.
- ❑ Приклади інтеграції Jenkins з іншими інструментами:
 - *Jenkins i Jira*: ця інтеграція дозволяє автоматично створювати завдання Jira для кожної збірки Jenkins.
 - *Jenkins i Slack*: ця інтеграція дозволяє надсилати сповіщення в Slack про результати збірок Jenkins.
 - *Jenkins i інші засоби комунікації*: Jenkins можна інтегрувати з іншими засобами комунікації, такими як Confluence, HipChat та іншими.

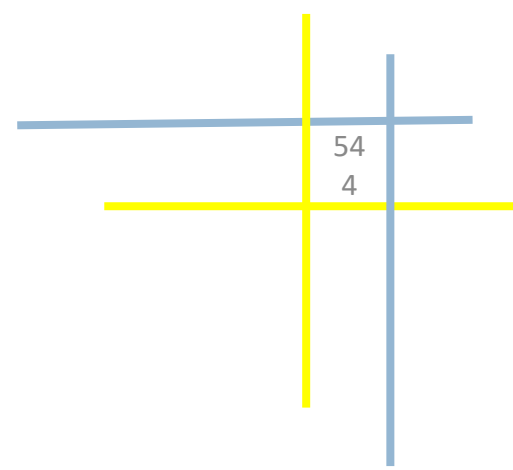


Захист Jenkins від несанкціонованого доступу

543

❑ Налаштування безпеки Jenkins:

- *Встановлення пароля для Jenkins*: цей параметр дозволяє захистити Jenkins від несанкціонованого доступу через веб-інтерфейс.
- *Використання SSL/TLS*: цей параметр дозволяє шифрувати зв'язок між клієнтом і Jenkins.
- *Налаштування прав доступу*: цей параметр дозволяє контролювати, які користувачі мають доступ до Jenkins і які дії вони можуть виконувати.



Конфігурація та робота зі сховищем коду



Налаштування нового проекту

☐ Створення нового проекту:

- Перейдіть до сторінки "*New Item*".
- Виберіть тип проекту.
- Введіть назву проекту та інші необхідні відомості.
- Натисніть кнопку "*Create*".

☐ Типи проектів:

- *Freestyle Project*: цей тип проекту є найпростішим і дозволяє вам налаштувати проект вручну.
- *Pipeline Project*: цей тип проекту використовує Jenkins Pipeline для автоматизації завдань збірки та розгортання.
- *Multibranch Pipeline Project*: цей тип проекту дозволяє вам запускати Pipeline для кожного гіпотеза в системі управління джерелами.



Основні кроки конфігурації₁

- ❑ *Налаштування систем управління джерелами:* Якщо необхідно автоматизувати складання коду, потрібно налаштувати інтеграцію Jenkins з системою управління джерелами.
- ❑ *Налаштування завдань збірки:* можна налаштувати завдання збірки, щоб Jenkins виконував різні завдання, наприклад, компіляцію коду, запуск тестів та розгортання коду.
- ❑ *Налаштування розкладу запуску:* можна налаштувати розклад запуску, щоб Jenkins запускав завдання збірки за розкладом.



Основні кроки конфігурації₂

☐ *Налаштування систем управління джерелами:*

- Встановіть плагін для системи управління джерелами, з якою ви хочете інтегрувати Jenkins.
- Налаштуйте параметри інтеграції для системи управління джерелами.

☐ *Налаштування завдань збірки:*

- Використовуйте плагіни для додавання підтримки різних мов програмування та інших технологій.
- Налаштуйте параметри завдань збірки, щоб вони відповідали вашим потребам.

☐ *Налаштування розкладу запуску:*

- Використовуйте плагіни для створення складних розкладів запуску.
- Налаштуйте параметри розкладу запуску, щоб вони відповідали вашим потребам.



Робота з залежностями

□ Методи управління залежностями:

- *Управління залежностями в коді:* цей метод полягає в маніпулюванні залежностями безпосередньо в коді.
- *Управління залежностями за допомогою менеджера пакетів:* цей метод полягає в використанні менеджера пакетів для управління залежностями.
- *Управління залежностями за допомогою контейнерів:* цей метод полягає в використанні контейнерів для управління залежностями.



Плагіни Jenkins₁

- ❑ Плагіни - це розширення, які можна використовувати для додавання нових функцій і можливостей до Jenkins.
- ❑ Jenkins підтримує тисячі плагінів, які можна використовувати для різних цілей.
- ❑ Важливі плагіни для розширення можливостей Jenkins:
 - `Jenkins Pipeline`: цей плагін дозволяє використовувати Jenkins Pipeline для автоматизації завдань збірки та розгортання.
 - `JUnit`: цей плагін дозволяє інтегрувати Jenkins з інструментом тестування JUnit.
 - `Selenium`: цей плагін дозволяє інтегрувати Jenkins з інструментом тестування Selenium.
 - `Cucumber`: цей плагін дозволяє інтегрувати Jenkins з інструментом тестування Cucumber.

JUnit



Cucumber



Плагіни Jenkins₂

550

❑ Інші важливі плагіни:

- `Artifactory`: цей плагін дозволяє Jenkins зберігати артефакти збірки в `Artifactory`.
- `SonarQube`: цей плагін дозволяє Jenkins виконувати статичний аналіз коду за допомогою `SonarQube`.
- `Slack`: цей плагін дозволяє Jenkins надсилати сповіщення в `Slack`.
- `GitHub`: цей плагін дозволяє Jenkins інтегруватися з `GitHub`.



sonarqube

❑ Додаткові деталі:

- Плагіни можна встановити з `Jenkins Update Center` або з локального файлу.
- Плагіни можна налаштувати, щоб вони відповідали вашим потребам.





Робота зі сховищем коду₁

- ❑ Jenkins може бути інтегрований з різними системами контролю версій, такими як Git, SVN та іншими. Це дозволяє Jenkins автоматично отримувати доступ до коду вашого проекту з системи контролю версій і запускати завдання збірки кожного разу, коли в репозиторій вносяться зміни.
- ❑ Інтеграція з системами контролю версій:
 - *Встановлення плагіну для системи контролю версій:* потрібно встановити плагін для системи контролю версій, з якою потрібно інтегрувати Jenkins.
 - *Налаштування параметрів інтеграції:* після встановлення плагінів необхідно налаштувати параметри інтеграції для кожної системи контролю версій.



Робота зі сховищем коду₂

- ❑ Jenkins підтримує Git за допомогою плагіна `Git Plugin`. Цей плагін дозволяє Jenkins отримувати доступ до репозиторіїв Git, таких як GitHub, Bitbucket та інші.
- ❑ Jenkins підтримує SVN за допомогою плагіна `Subversion Plugin`. Цей плагін дозволяє Jenkins отримувати доступ до репозиторіїв SVN.



git



Робота зі сховищем коду₃

❑ Налаштування параметрів інтеграції для систем контролю версій:

- URL репозиторію
- Ім'я користувача
- Пароль
- Інші параметри, такі як галузь, яка повинна використовуватися, та інші



Робота зі сховищем коду₄

- ☐ Після того, як інтегровано Jenkins з системою контролю версій, можна налаштувати завдання збірки, щоб Jenkins запускав завдання збірки кожного разу, коли в репозиторій вносяться зміни.
- ☐ Можна використовувати плагіни для розширення функціональності інтеграції з системами контролю версій.



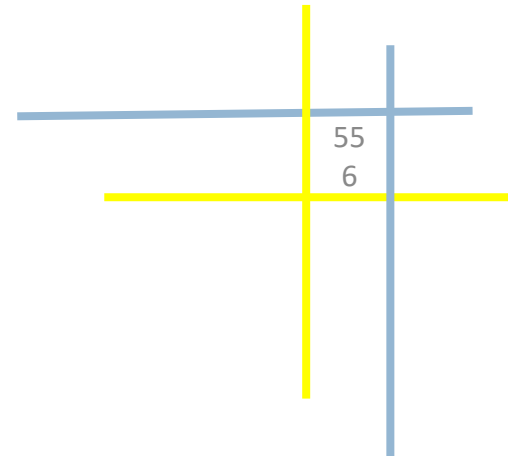
Робота зі сховищем коду₅

❑ Підтримка Git:

- Jenkins може отримувати доступ до репозиторіїв Git за допомогою різних протоколів, таких як SSH, HTTP та HTTPS.
- Jenkins може використовувати Git для виконання різних завдань, таких як компіляція коду, запуск тестів та розгортання коду.

❑ Підтримка SVN:

- Jenkins може отримувати доступ до репозиторіїв SVN за допомогою протоколу HTTP.
- Jenkins може використовувати SVN для виконання різних завдань, таких як компіляція коду, запуск тестів та розгортання коду.



Автоматична збірка та тестування



Налаштування автоматичної збірки₁

❑ Для настройки автоматичної збірки в Jenkins необхідно виконати такі кроки:

- Створити проект Jenkins.
- Налаштувати інтеграцію з системою управління версіями.
- Налаштувати завдання збірки.

❑ Створення проекту Jenkins:

1. Перейдіть до сторінки "New Item".
2. Виберіть тип проекту "Freestyle Project".
3. Введіть назву проекту та інші необхідні відомості.
4. Натисніть кнопку "Create".



Налаштування автоматичної збірки₂

☐ Налаштування інтеграції з системою управління версіями:

1. Виберіть вкладку "Build".
2. У розділі "Trigger Build" виберіть опцію "Poll SCM".
3. Введіть URL репозиторію та інші необхідні відомості.

☐ Налаштування завдання збірки:

1. Виберіть вкладку "Build".
2. У розділі "Steps" додайте кроки, які потрібно виконати для збірки коду.
3. Натисніть кнопку "Save".

☐ Кроки збірки:

- * **Компіляція коду:** Цей крок компілюється код проекту.
- * **Запуск тестів:** Цей крок запускає тести проекту.
- * **Розгортання коду:** Цей крок розгортає код проекту в виробниче середовище.



Оцінка результатів збірки

□ Інструменти для аналізу результатів:

- SonarQube: цей інструмент використовується для виконання статичного аналізу коду.
- Checkstyle: цей інструмент використовується для перевірки коду на дотримання певних правил.
- PMD: цей інструмент використовується для виявлення потенційних проблем у коді.
- FindBugs: цей інструмент використовується для виявлення помилок у коді.



SonarQube

- ☐ Цей інструмент можна використовувати для оцінки якості коду за допомогою таких критеріїв, як читабельність, зрозумілість та безпека.
- ☐ SonarQube можна використовувати для виявлення потенційних проблем у коді, таких як помилки та недоліки.



Checkstyle

- ☐ Цей інструмент можна використовувати для перевірки коду на дотримання певних правил.
- ☐ Checkstyle можна використовувати для виявлення потенційних проблем у коді, таких як помилки форматування та кодування.



PMD та FindBugs

□PMD:

- Цей інструмент можна використовувати для виявлення потенційних проблем у коді, таких як дублювання коду, потенційні помилки та недоліки.

□FindBugs:

- Цей інструмент можна використовувати для виявлення помилок у коді, таких як помилки пам'яті, помилки безпеки та інші помилки.



Рекомендації та стратегії для оптимізації збірки

- ☐ *Використання паралельних завдань*: цей метод дозволяє виконувати завдання збірки одночасно, що може значно скоротити час виконання збірки.
- ☐ *Використання кешування*: цей метод дозволяє зберігати результати завдань збірки, щоб не потрібно було виконувати їх повторно.
- ☐ *Використання конвеєра завдань*: цей метод дозволяє виконувати завдання збірки в певному порядку, що може допомогти оптимізувати використання ресурсів.
- ☐ *Використання хмарних технологій*: цей метод дозволяє розгорнути Jenkins на хмарі, що може забезпечити більшу пропускну здатність і продуктивність.



Налаштування тестування₁

- ❑ Автоматизація тестування - це практика, яка полягає в тому, щоб автоматизувати виконання тестів. Це дозволяє виконувати тести швидко та ефективно, а також допомагає виявляти помилки в коді на ранніх етапах розробки.
- ❑ Jenkins підтримує багато різних інструментів тестування, таких як JUnit, Selenium, Cucumber та інші.
- ❑ Для настройки інтеграції інструментів тестування з Jenkins необхідно виконати такі кроки:
 - Встановіть плагін для інструменту тестування, який необхідно використовувати.
 - Налаштуйте параметри плагіна, щоб він відповідав визначеним потребам.



Налаштування тестування₂

❑ Встановлення плагіна для інструменту тестування:

1. Перейдіть до сторінки "Manage Plugins".
2. У розділі "Available" знайдіть плагін для інструменту тестування, який ви хочете встановити.
3. Натисніть кнопку "Install".

❑ Налаштування параметрів плагіна:

1. Перейдіть до сторінки налаштувань плагіна.
2. Введіть необхідні параметри.
3. Натисніть кнопку "Save".



Налаштування тестування₃

□ Приклади інструментів тестування, які можна інтегрувати з Jenkins:

- `JUnit`: цей інструмент використовується для тестування Java-коду.
- `Selenium`: цей інструмент використовується для тестування веб-застосунків.
- `Cucumber`: цей інструмент використовується для написання тестів, які використовують природну мову.

□ Інші інструменти тестування, які можна інтегрувати з Jenkins:

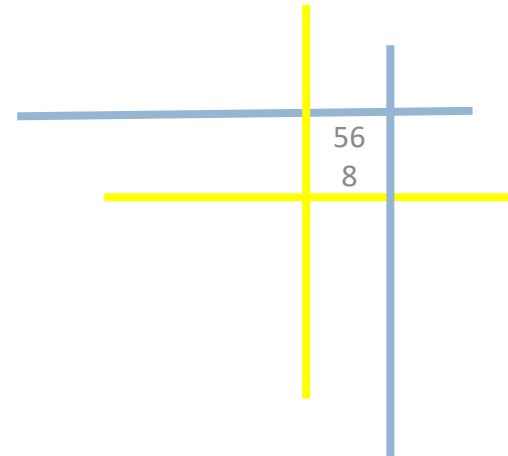
- `TestNG`: цей інструмент використовується для тестування Java-коду.
- `Appium`: цей інструмент використовується для тестування мобільних застосунків.
- `Robot Framework`: цей інструмент використовується для написання тестів, які використовують сценарії.



Планування збірки в Jenkins

□ Jenkins надає кілька способів планування збірки:

- *Розклади збірок*: Jenkins дозволяє створювати розклади збірок, які запускають збірку в певний час або день.
- *Trigger-и збірок*: Jenkins дозволяє використовувати trigger-и збірок для запуску збірки, коли відбувається певна подія, наприклад, зміна коду або випуск нових версій бібліотек.
- *Build Pipeline*: Jenkins Pipeline дозволяє автоматизувати планування збірки за допомогою конвеєра завдань.



Моніторинг та логування



Відстеження стану проектів₁

- ❑ Jenkins надає засоби для моніторингу та аналізу логів, які можна використовувати для відстеження стану проектів.

- ❑ Засоби моніторингу Jenkins:
 - *Jenkins Dashboard*: цей інструмент надає загальний огляд стану всіх проектів Jenkins.
 - *Jenkins Build History*: цей інструмент дозволяє переглядати історію збірок для кожного проекту.
 - *Jenkins Console Output*: цей інструмент дозволяє переглядати консольний вивід для кожного завдання збірки.



Відстеження стану проектів₂

❑ Засоби аналізу логів Jenkins:

- *Jenkins Logstash Plugin*: цей плагін дозволяє інтегрувати Jenkins з Logstash для аналізу логів.
- *Jenkins ELK Plugin*: цей плагін дозволяє інтегрувати Jenkins з ELK для аналізу логів.



Jenkins Dashboard

❑ Цей інструмент можна використовувати для відстеження таких показників, як:

- Статус збірки
- Час збірки
- Кількість помилок
- Кількість успішних збірок

Jenkins

PipelinesAdministration

Logout

PipelinesSearch pipelines...New Pipeline

Favorites

×

building-a-multibranch-pipeline-project

development

#2ec7747

a minute ago

↺▶★

—

simple-node-js-react-npm-app

master

#be4ffd6

31 minutes ago

↺▶★

!

simple-java-maven-app

master

#e51db83

24 minutes ago

↺▶★

○

building-a-multibranch-pipeline-project

master

#87393ab

a few seconds ago

▶★

✓

building-a-multibranch-pipeline-project

production

#e85d115

7 minutes ago

↺▶★



Jenkins Build History

□ Цей інструмент можна використовувати для перегляду таких даних, як:

- Результат збірки
- Вихідні дані збірки
- Консольний вивід збірки

The screenshot displays the Jenkins web interface for the 'beaglebone' node. The main section is titled 'Build History on beaglebone'. It features a calendar view showing builds from April 4 to April 9. Below the calendar, a table lists the build details:

Build	Time Since ↑	Status
beaglebone.default.Benchmark.reboot #4	27 min	aborted
beaglebone.default.Functional.netperf #3	33 min	aborted
beaglebone.default.Benchmark.reboot #3	33 min	aborted
beaglebone.default.Benchmark.dbench #4	33 min	aborted
beaglebone.default.Benchmark.interbench #3	38 min	aborted



Jenkins Console Output

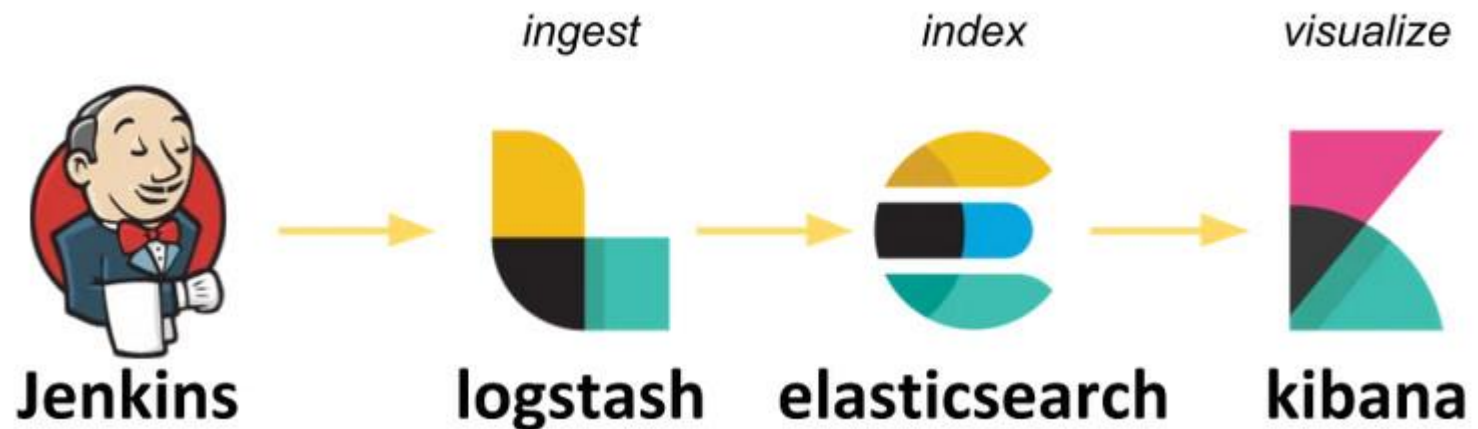
- ❑ Цей інструмент можна використовувати для перегляду консольного виводу для кожного завдання збірки

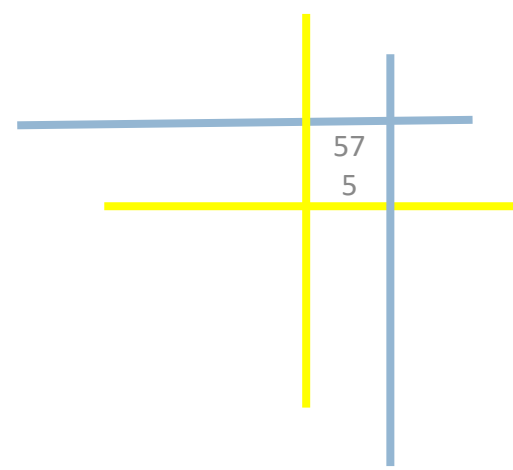
The screenshot shows the Jenkins web interface. At the top, there's a black header with the Jenkins logo and name. Below it, a breadcrumb trail shows 'Jenkins' > 'raspberryPi3TakePicture' > '#27'. On the left sidebar, there are several links: 'Back to Project' (green up arrow), 'Status' (magnifying glass), 'Changes' (pencil), 'Console Output' (terminal icon, highlighted in purple), 'View as plain text' (document icon), 'Edit Build Information' (pencil), 'Delete build '#27'' (red circle with slash), 'Parameters' (document icon), 'Previous Build' (green left arrow), and 'Next Build' (green right arrow). The main content area is titled 'Console Output' with a blue sphere icon. It displays the following text: 'Started by remote host 0:0:0:0:0:0:1', 'Running as SYSTEM', 'Building remotely on [Node_01_RP](#) in workspace /home/pi/workspace/raspberrypi3TakePicture', '[raspberryPi3TakePicture] \$ /bin/sh -xe /tmp/jenkins852966680965527208.sh', '+ python --version', 'Python 2.7.13', '+ python /home/pi/Desktop/camera.py', 'done', and 'Finished: SUCCESS'.



Jenkins Logstash Plugin

- ❑ Цей плагін дозволяє інтегрувати Jenkins з ELK для аналізу логів.
- ❑ ELK - це складання програмного забезпечення для керування логами, яке складається з Elasticsearch, Logstash і Kibana.





Використання контейнерів в Jenkins



Використання контейнерів в Jenkins

- ❑ Jenkins можна використовувати для інтеграції з контейнеризаційними технологіями, такими як Docker, для створення робочого середовища для збірки та тестування програмного забезпечення.
- ❑ Переваги використання контейнерів в Jenkins:
 - Контейнери забезпечують уніфіковане робоче середовище для всіх збірок.
 - Контейнери допомагають підвищити продуктивність і масштабованість збірок.
 - Контейнери допомагають зменшити час і складність розгортання.



Інтеграція Jenkins з Docker

- ❑ Jenkins надає кілька способів інтеграції з Docker.
 - Docker Plugin: цей плагін дозволяє Jenkins керувати контейнерами Docker.
 - Docker Pipeline: Jenkins Pipeline дозволяє використовувати Docker для створення і запуску контейнерів.
 - Docker Cloud: цей сервіс дозволяє Jenkins запускати контейнери Docker у хмарі.



Docker Plugin

- ☐ Цей плагін дозволяє Jenkins керувати контейнерами Docker.
- ☐ Цей плагін також дозволяє Jenkins використовувати Docker для створення і запуску контейнерів під час збірки.



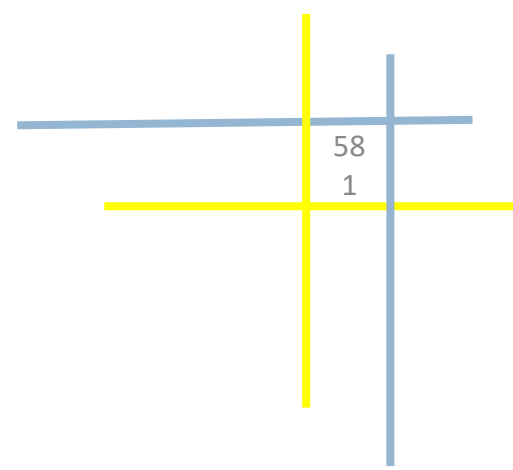
Docker Pipeline

- ☐ Jenkins Pipeline дозволяє використовувати Docker для створення і запуску контейнерів.
- ☐ Це можна зробити, використовуючи спеціальні дії Docker, такі як `docker.image()`, `docker.build()` і `docker.run()`.



Docker Cloud

- ☐ Цей сервіс дозволяє Jenkins запускати контейнери Docker у хмарі.
- ☐ Це може бути корисно для масштабування збірок або для запуску збірок у середовищі, що не є локальним.



Масштабування Jenkins



Масштабування Jenkins

- ❑ Масштабування - це процес збільшення або зменшення розміру системи, щоб відповідати зростаючим або зменшуючим потребам.
- ❑ Jenkins можна масштабувати, щоб відповідати зростаючим потребам у продуктивності та пропускній здатності.



Горизонтальне масштабування

- ❑ Горизонтальне масштабування - це процес додавання додаткових вузлів до системи.
- ❑ Jenkins можна масштабувати горизонтально, додаючи додаткові агенти.
 - Це простий спосіб масштабувати Jenkins.
 - Однак, це може бути дорогим способом масштабування, оскільки необхідно придбати додаткове обладнання.



Вертикальне масштабування

- ❑ Вертикальне масштабування - це процес підвищення продуктивності та пропускної здатності окремих вузлів системи.
- ❑ Jenkins можна масштабувати вертикально, оновлюючи обладнання, на якому він працює.
 - Це більш ефективний спосіб масштабувати Jenkins, ніж горизонтальне масштабування.
 - Однак, це може бути складним способом масштабування, оскільки необхідно оновлювати обладнання.



Інші способи масштабування Jenkins

❑ Використання хмарних платформ

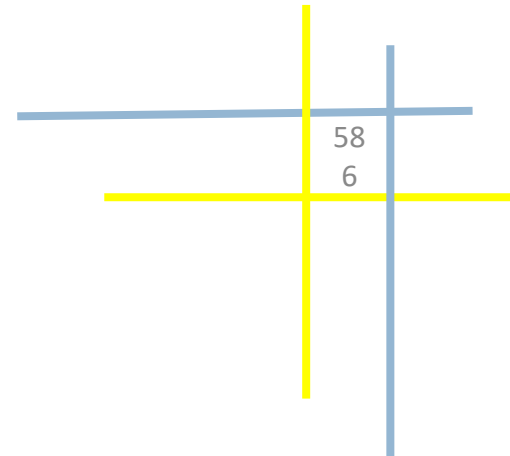
- Хмарні платформи, такі як Amazon Web Services, пропонують масштабування як послугу.
- Це може бути простим і ефективним способом масштабувати Jenkins.

❑ Використання контейнерів

- Контейнери дозволяють запускати Jenkins на будь-якому комп'ютері, який підтримує Docker.
- Це може бути корисним для масштабування Jenkins у хмарі або в розподіленому середовищі.

❑ Використання менеджера завдань

- Менеджери завдань, такі як Jenkins Pipeline, дозволяють виконувати завдання збірки паралельно.
- Це може допомогти підвищити продуктивність збірки.



Інтеграція з засобами інфраструктури



Інтеграція з інструментами для автоматизації інфраструктури

- ❑ Інфраструктурні засоби - це інструменти, які використовуються для управління та автоматизації інфраструктури.
- ❑ Інтеграція Jenkins з інфраструктурними засобами дозволяє автоматизувати завдання інфраструктури, такі як розгортання, масштабування та відновлення.
- ❑ Переваги інтеграції Jenkins з інфраструктурними засобами:
 - *Підвищена продуктивність*: інтеграція Jenkins з інфраструктурними засобами може допомогти підвищити продуктивність, автоматизуючи завдання інфраструктури.
 - *Підвищена точність*: інтеграція Jenkins з інфраструктурними засобами може допомогти підвищити точність, забезпечуючи, що завдання інфраструктури виконуються правильно.
 - *Підвищена повторюваність*: інтеграція Jenkins з інфраструктурними засобами може допомогти підвищити повторюваність, забезпечуючи, що завдання інфраструктури виконуються однаково кожен раз.



Типи інфраструктурних засобів, які можна інтегрувати з Jenkins

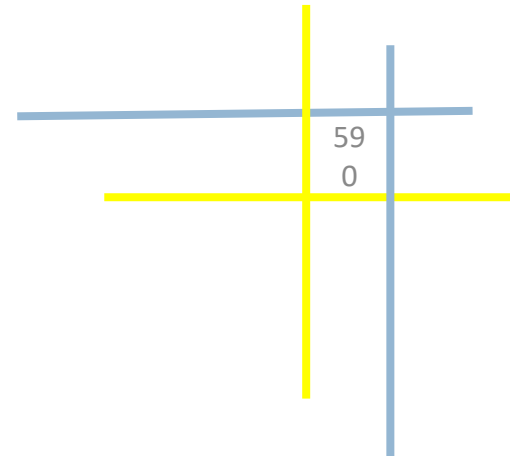
- ☐ Інструменти управління інфраструктурою, такі як Terraform, Ansible та Puppet.
- ☐ Інструменти управління хмарними ресурсами, такі як AWS CloudFormation, Azure Resource Manager та Google Cloud Deployment Manager.
- ☐ Інструменти управління конфігураціями, такі як Chef, SaltStack та Puppet.



Інтеграція Jenkins з інфраструктурними засобами

□ Jenkins надає кілька способів інтеграції з інфраструктурними засобами.

- Плагіни Jenkins, такі як Terraform Plugin, Ansible Plugin та Puppet Plugin.
- Конвеєр завдань Jenkins, який можна використовувати для автоматизації завдань інфраструктури.
- Інтеграція Jenkins з хмарної платформи, яка може включати інтеграцію з інфраструктурними засобами, які підтримуються цією платформою.



Реальний приклад



Реальний приклад₁

- ❑ Компанія розробляє веб-застосунок, який працює в хмарі. Компанія використовує Jenkins для автоматизації збірки, тестування та розгортання додатка.
- ❑ Конвеєр завдань Jenkins виконує такі дії:
 - Збирає код застосунка з репозиторію Git.
 - Будує застосунок за допомогою Maven.
 - Запускає тести застосунка за допомогою JUnit.
 - Розгортає застосунок на веб-хостинг.



Реальний приклад₂

- ❑ Щоб уніфікувати робоче середовище для збірки та тестування, компанія використовує Vagrant для створення віртуальної машини. Vagrant дозволяє швидко і легко створювати і керувати віртуальними машинами.
- ❑ Щоб забезпечити повторюваність і масштабованість збірки та тестування, компанія використовує Docker для створення контейнерів. Docker дозволяє запускати програми в ізольованому середовищі, що забезпечує повторюваність і масштабованість.
- ❑ Щоб автоматизувати налаштування віртуальної машини та контейнерів, компанія використовує Ansible. Ansible - це інструмент для автоматизації конфігурації, який дозволяє створювати скрипти для налаштування комп'ютерів.



Реальний приклад₃

❑ Конвеєр завдань Jenkins виконує такі дії:

- Використовує Vagrant для створення віртуальної машини.
- За допомогою Ansible налаштовує віртуальну машину.
- За допомогою Docker створює контейнери для збірки та тестування.
- За допомогою Maven збирає застосунок.
- За допомогою JUnit запускає тести застосунку.
- За допомогою Ansible розгортає застосунок на веб-хостинг.

Донецький національний технічний університет



Тема 8. Системи оркестрації.

Лекція 13. Вступ до Kubernetes

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

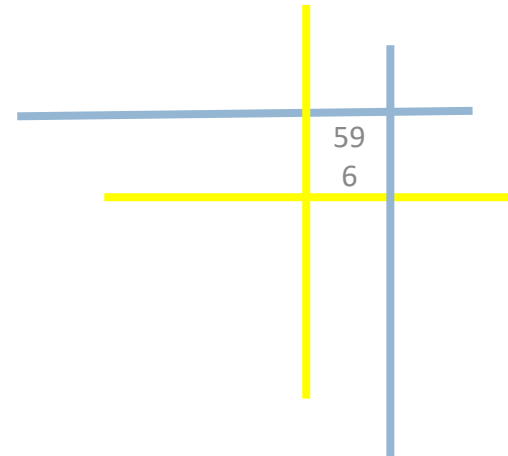
Укладач: проф. Дорогий Я.Ю.



Лекційні питання

595

- ☐ Вступ до Kubernetes
- ☐ Архітектура Kubernetes
- ☐ Ресурси Kubernetes



Вступ до Kubernetes



Вступ до Kubernetes

- ❑ Kubernetes - це система контейнеризації та управління, яка використовується для автоматизації розгортання, масштабування та управління контейнеризованими застосунками. Він використовується багатьма компаніями, включаючи Google, Netflix і Spotify.
- ❑ Kubernetes був розроблений компанією Google в 2014 році. Він був відкритий у 2015 році і швидко став популярним серед розробників та операційних команд.





Створення Kubernetes

- Деякі важливі пункти створення Kubernetes включають:
- *Контейнеризація*: Kubernetes був розроблений з урахуванням контейнеризації. Він забезпечує підтримку для популярних контейнерних платформ, таких як Docker і CRI-O.
 - *Автоматизація*: Kubernetes автоматизує розгортання, масштабування та управління контейнеризованими застосунками. Це дозволяє розробникам та операційним командам швидше і простіше розгортати та масштабувати застосунки.
 - *Еластичність*: Kubernetes забезпечує еластичність, дозволяючи застосункам масштабуватися вгору або вниз відповідно до навантаження.
 - *Надійність*: Kubernetes забезпечує надійність, дозволяючи застосункам залишатися доступними навіть у разі несправності окремих вузлів кластера.



Значення Kubernetes у сучасній розробці

- ❑ Kubernetes став важливим компонентом сучасної розробки. Він дозволяє розробникам швидше і простіше розгортати та масштабувати контейнеризовані застосунки.
- ❑ Kubernetes забезпечує ряд переваг, включаючи:
 - Автоматизацію розгортання та масштабування
 - Збільшення надійності та доступності
 - Покращення безпеки
 - Зменшення витрат



Вигоди використання Kubernetes

❑ Масштабованість

- Kubernetes забезпечує еластичність, дозволяючи додаткам масштабуватися вгору або вниз відповідно до навантаження. Це дозволяє розробникам та операційним командам швидко і легко реагувати на зміни попиту.

❑ Управління ресурсами

- Kubernetes допомагає керувати ресурсами, використовуючи різні стратегії розгортання та масштабування. Це дозволяє оптимізувати використання ресурсів і зменшити витрати.

❑ Висока доступність

- Kubernetes забезпечує високу доступність, дозволяючи застосункам залишатися доступними навіть у разі несправності окремих вузлів кластера. Це досягається за допомогою таких функцій, як балансування навантаження, автоматичне відновлення та відновлення після збоїв.



Масштабованість

❑ Kubernetes забезпечує еластичність, дозволяючи застосункам масштабуватися вгору або вниз відповідно до навантаження. Це досягається за допомогою таких функцій, як:

- *Розгортання з повторюваністю*: Kubernetes дозволяє розробникам розгортати додатки в кластері з повторюваністю. Це означає, що додатки можна масштабувати, просто додаючи або видаляючи робочі процеси.
- *Балансування навантаження*: Kubernetes забезпечує балансування навантаження між робочими процесами. Це допомагає забезпечити, щоб всі робочі процеси отримували однакову кількість навантаження.
- *Автоматичне масштабування*: Kubernetes дозволяє автоматично масштабувати додатки відповідно до навантаження. Це можна зробити за допомогою різних стратегій масштабування, таких як масштабування за потребою або масштабування за прогнозованим навантаженням.



Управління ресурсами

- ❑ Kubernetes допомагає керувати ресурсами, використовуючи різні стратегії розгортання та масштабування. Це дозволяє оптимізувати використання ресурсів і зменшити витрати.
- ❑ Деякі з функцій Kubernetes, які допомагають керувати ресурсами, включають:
 - *Оптимізація використання процесорів:* Kubernetes може оптимізувати використання процесорів, розподіляючи робочі процеси між вузлами кластера.
 - *Оптимізація використання пам'яті:* Kubernetes може оптимізувати використання пам'яті, видаляючи робочі процеси, які не використовують пам'ять.
 - *Оптимізація використання мережі:* Kubernetes може оптимізувати використання мережі, розподіляючи запити до служб між вузлами кластера.



Висока доступність

❑ Kubernetes забезпечує високу доступність, дозволяючи додаткам залишатися доступними навіть у разі несправності окремих вузлів кластера. Це досягається за допомогою таких функцій, як:

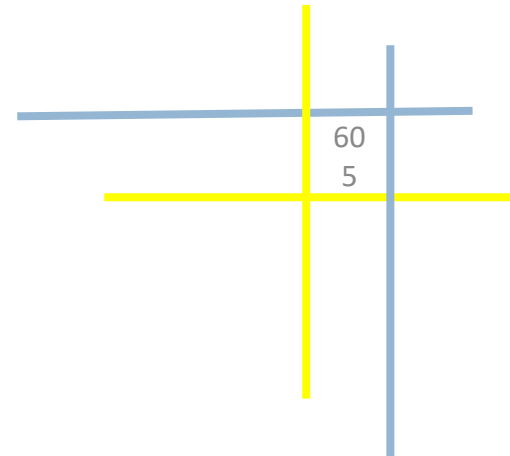
- *Автоматичне відновлення*: Kubernetes автоматично перезапускає робочі процеси, які виходять з ладу.
- *Відновлення після збоїв*: Kubernetes зберігає стан робочих процесів, щоб їх можна було відновити після збоїв.
- *Балансування навантаження*: Kubernetes забезпечує балансування навантаження між робочими процесами, навіть якщо деякі з них недоступні.



Основні поняття та визначення

❑ Основні поняття та визначення Kubernetes включають:

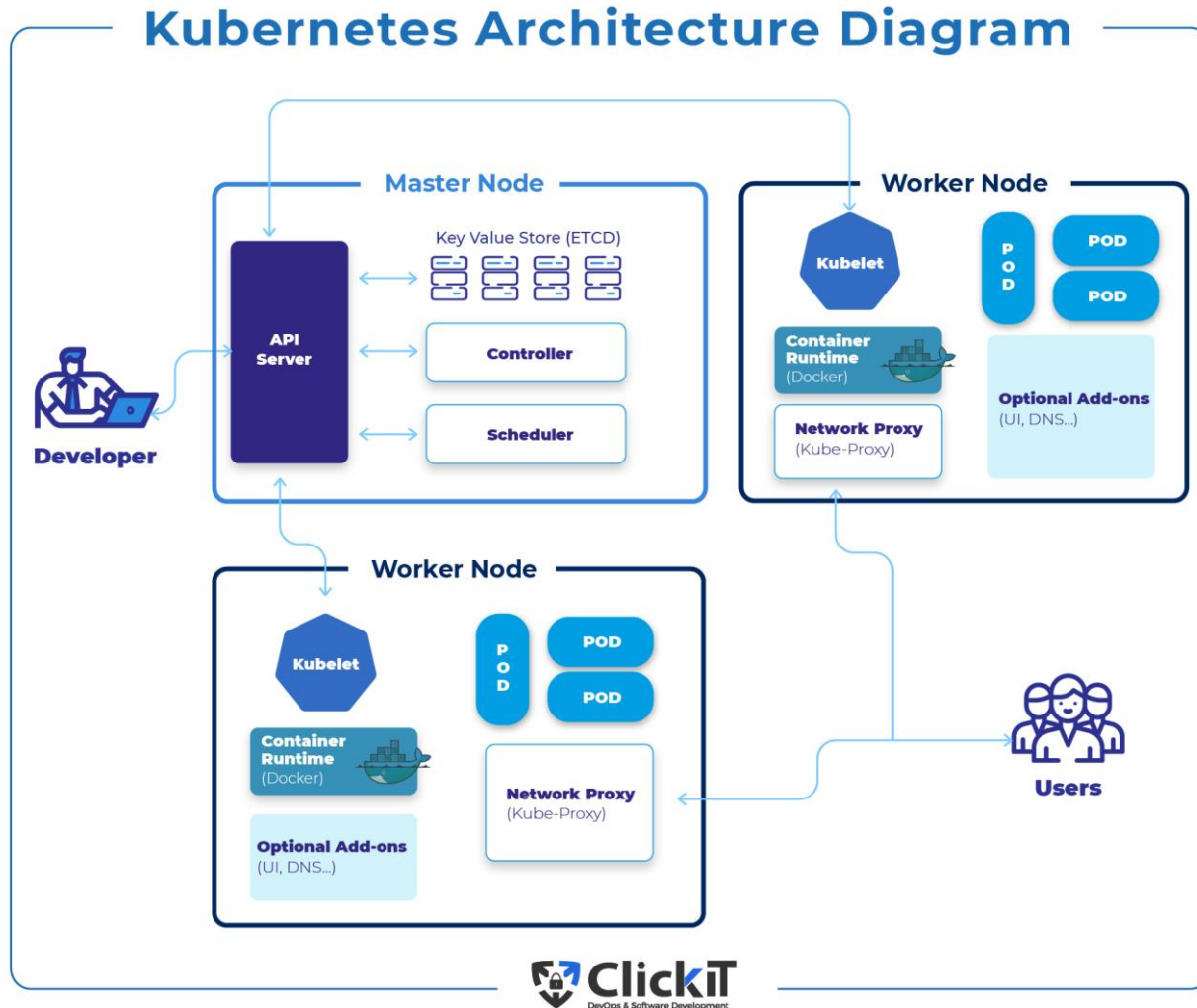
- *Контейнер*: невеликий, ізольований процес, який містить у собі все необхідне для запуску програми, включаючи код, бібліотеки та конфігурацію.
- *Кластер*: група фізичних або віртуальних машин, які працюють разом, щоб розгортати та керувати контейнерами.
- *Підрозділи*: набори контейнерів, які працюють разом, щоб виконувати певну задачу.
- *Робота*: одиниця розгортання та масштабування контейнеризованого застосунку.
- *Служба*: екземпляр роботи, який доступний через мережу.



Архітектура Kubernetes



Архітектура Kubernetes





Архітектурні принципи Kubernetes₁

- ❑ Мастер-ноди є центральним компонентом кластера Kubernetes. Вони відповідають за такі завдання, як:
 - Управління вузлами
 - Розгортання і масштабування контейнерів
 - Балансування навантаження
 - Автоматичне відновлення
 - Відновлення після збоїв



Архітектурні принципи Kubernetes₂

- ❑ Вузли є фізичними або віртуальними машинами, які працюють в кластері Kubernetes. Вони відповідають за такі завдання, як:
- Розгортання контейнерів
 - Виконання контейнерів
 - Зв'язок між контейнерами
 - Управління мережею



Робочі процеси Kubernetes₁

□ Створення та управління контейнерами

- Kubernetes використовує ресурси, відомі як контейнерні образи, для створення контейнерів. Контейнерні образи - це набори файлів, які містять все необхідне для запуску контейнера, включаючи код, бібліотеки та конфігурацію.
- Kubernetes дозволяє розробникам та операційним командам створювати та керувати контейнерами за допомогою API Kubernetes. API Kubernetes дозволяє розробникам визначати, які контейнери потрібно розгорнути, на яких вузлах кластера і з якими ресурсами.



Робочі процеси Kubernetes₂

□ Оркестрація та автоматизація

- Kubernetes використовує процес, відомий як оркестрація, для розгортання та управління контейнерами в кластері. Оркестрація - це процес автоматизації завдань, пов'язаних з розгортанням, масштабуванням та управлінням контейнеризованими застосунками.
- Kubernetes використовує такі функції оркестрації, як:
 - *Розгортання*: Kubernetes дозволяє розробникам розгорнути контейнери в кластері за допомогою простих команд.
 - *Масштабування*: Kubernetes дозволяє розробникам масштабувати контейнеризовані застосунки вгору або вниз відповідно до навантаження.
 - *Автоматичне відновлення*: Kubernetes автоматично перезапускає робочі процеси, які виходять з ладу.
 - *Відновлення після збоїв*: Kubernetes зберігає стан робочих процесів, щоб їх можна було відновити після збоїв.



Створення та управління контейнерами

- ❑ Kubernetes використовує ресурси, відомі як контейнерні образи, для створення контейнерів. Контейнерні образи - це набори файлів, які містять все необхідне для запуску контейнера, включаючи код, бібліотеки та конфігурацію.
- ❑ Контейнерні образи можна створювати вручну або за допомогою інструментів, таких як Docker. Для створення контейнерного образу за допомогою Docker ви можете використовувати команду `docker build`.
- ❑ Після створення контейнерного образу ви можете розгорнути його в кластері Kubernetes за допомогою команди `kubectl apply`:
 - `kubectl apply -f my-deployment.yaml`
 - Ця команда розгорне контейнерний образ з ім'ям `my-image` у кластері Kubernetes.



Компоненти оркестрації Kubernetes.

Kubernetes API

- ❑ Kubernetes API - це RESTful API, який використовується для взаємодії з Kubernetes. API дозволяє розробникам і операційним командам створювати, керувати та налаштовувати кластери Kubernetes.
- ❑ API Kubernetes організований навколо ресурсів. Ресурси - це об'єкти, які описують компоненти кластера Kubernetes, такі як робочі процеси, служби та томи.
- ❑ API Kubernetes можна використовувати з будь-якої мови програмування або інструменту. Він доступний через HTTP, HTTPS і gRPC.



Компоненти оркестрації Kubernetes.

Еталонні контролери

- ❑ Еталонні контролери - це набори правил, які описують, як розгортати та керувати контейнеризованими застосунками. Kubernetes включає в себе кілька стандартних еталонних контролерів, таких як реплікаційні контролери та демон-контролери.
- ❑ Еталонні контролери можна використовувати для створення та управління контейнеризованими застосунками без необхідності кодувати власну програму оркестрації.



Реплікаційні контролери

- ❑ Реплікаційні контролери відповідають за розгортання і підтримку заданої кількості робочих процесів для кожного об'єкта. Наприклад, реплікаційний контролер може бути використаний для розгортання трьох робочих процесів веб-сервера.
- ❑ Реплікаційні контролери використовуються для забезпечення того, щоб застосунки завжди були доступними. Вони також можуть використовуватися для масштабування застосунків відповідно до навантаження.



Демон-контролери

- ❑ Демон-контролери відповідають за керування робочими процесами, які повинні працювати постійно. Наприклад, демон-контролер може бути використаний для запуску демонічного процесу, такого як процес аналізу даних.
- ❑ Демон-контролери використовуються для забезпечення того, щоб важливі процеси завжди працювали. Вони також можуть використовуватися для виконання завдань, які вимагають постійного моніторингу або обслуговування.



Роль Kubelet у системі

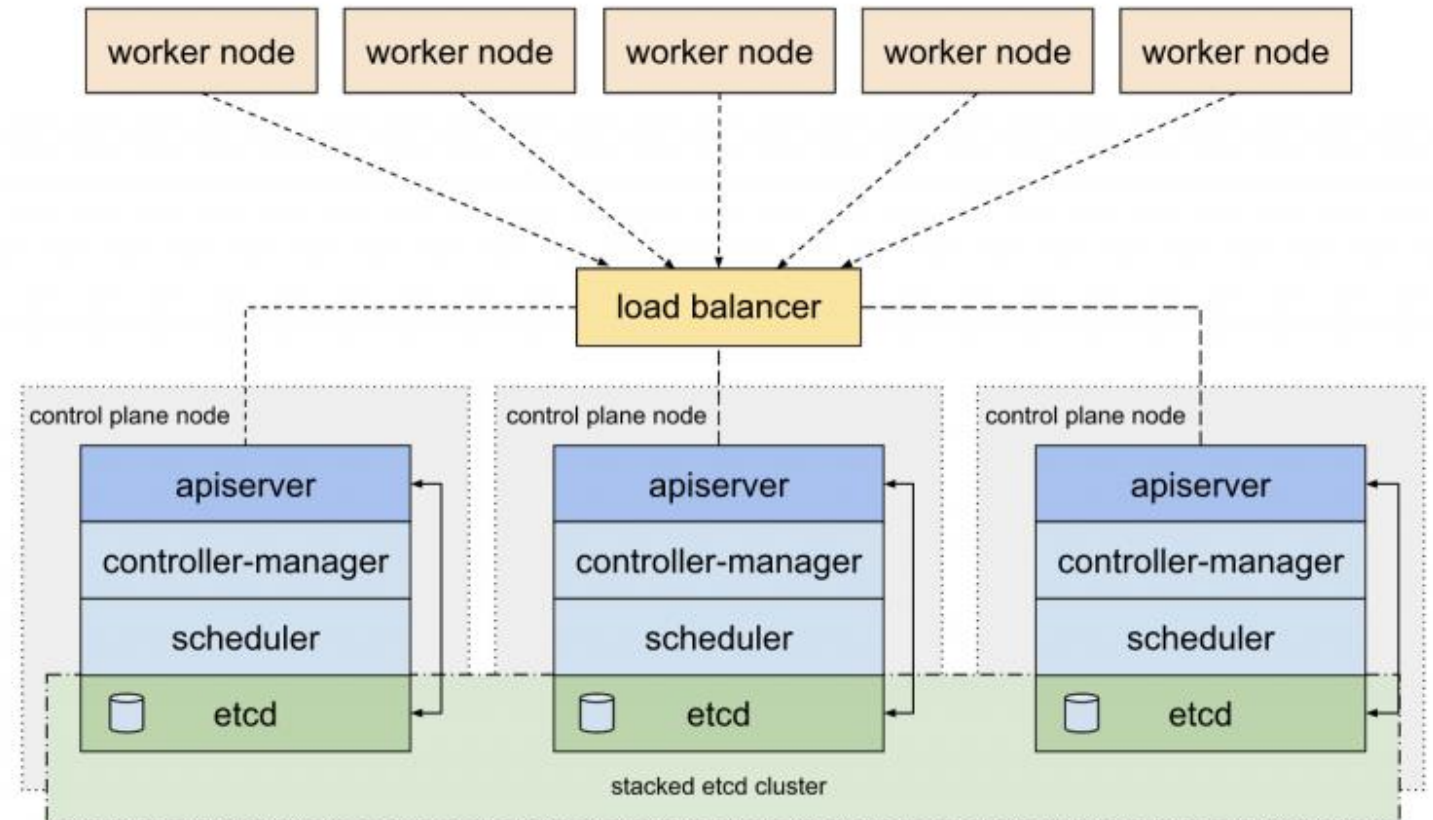
- ❑ Kubelet - це агент, який працює на кожному вузлі кластера Kubernetes. Kubelet відповідає за моніторинг та управління контейнерами на цьому вузлі.
- ❑ Kubelet виконує такі завдання:
 - Запуск і зупинка контейнерів
 - Перезапуск контейнерів, які вийшли з ладу
 - Управління ресурсами, які виділяються контейнерам
 - Підключення контейнерів до мережі
- ❑ Kubelet отримує завдання від API-сервера Kubernetes. API-сервер повідомляє Kubelet, які контейнери потрібно запустити, які ресурси виділити контейнерам і як підключити контейнери до мережі.



Мастер-сервер₁

□ Роль та функції мастер-сервера

- API-сервер
- Контролер-менеджер
- etcd





Мастер-сервер. API-сервер

- ❑ API-сервер - це центральний компонент Kubernetes. Він відповідає за взаємодію з користувачами та інструментами. API-сервер дозволяє розробникам і операційним командам створювати, керувати та налаштовувати кластери Kubernetes.
- ❑ API-сервер організований навколо ресурсів. Ресурси - це об'єкти, які описують компоненти кластера Kubernetes, такі як робочі процеси, служби та томи.
- ❑ API-сервер доступний через HTTP, HTTPS і gRPC.



Мастер-сервер. Контролер-менеджер

- ❑ Контролер-менеджер - це компонент, який відповідає за розгортання та управління контейнеризованими застосунками. Контролер-менеджер використовує контролери, щоб розгорнути і підтримати задану кількість робочих процесів для кожного об'єкта.
- ❑ Контролер-менеджер включає в себе такі стандартні контролери, як:
 - Реплікаційні контролери
 - Демон-контролери
 - Помилкові контролери
 - Службові контролери



Мастер-сервер. `etcd`

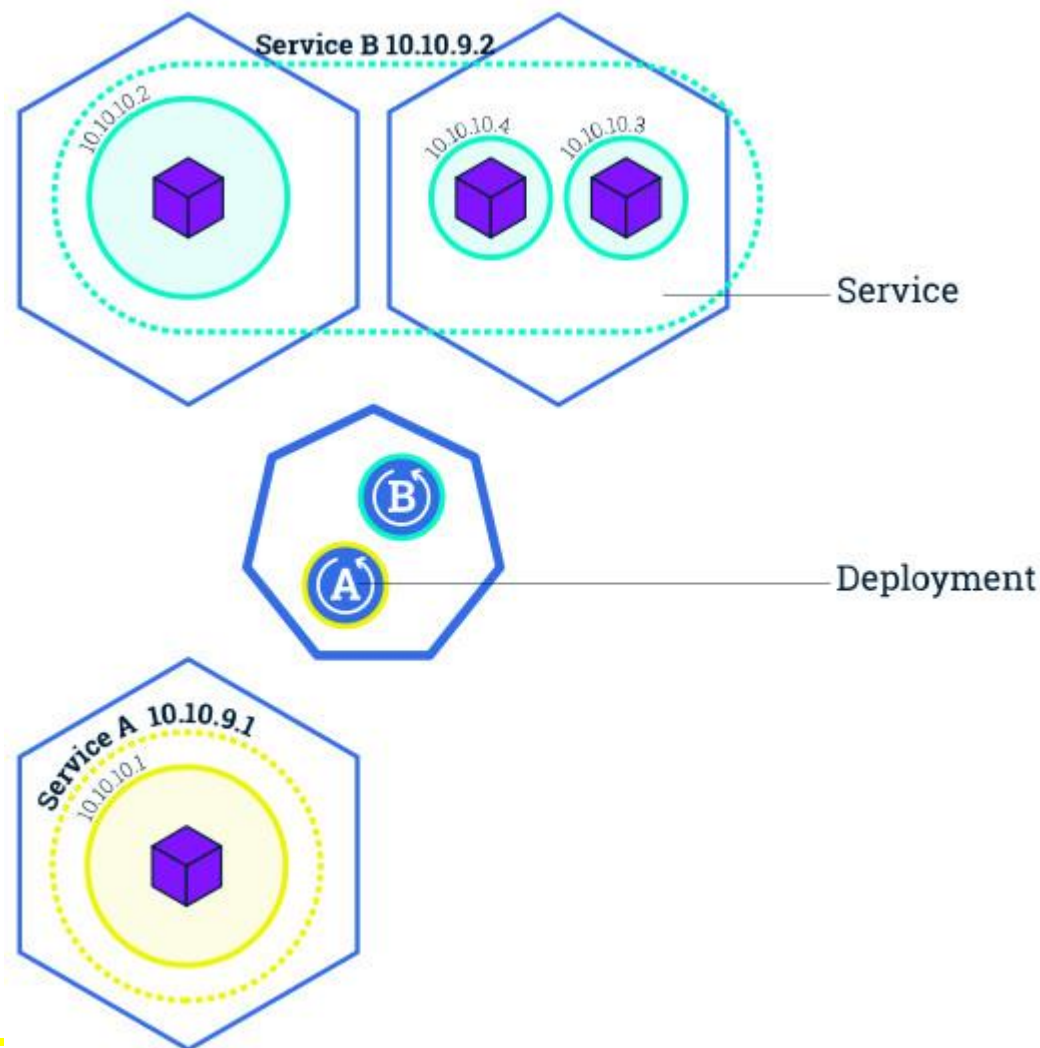
- ❑ `etcd` - це розподілена база даних, яка використовується для зберігання конфігурації кластера Kubernetes. `etcd` зберігає інформацію про такі речі, як робочі процеси, служби та мережі.
- ❑ `etcd` забезпечує надійність і доступність конфігурації кластера Kubernetes. Він синхронізує дані між усіма вузлами кластера, щоб гарантувати, що всі вузли завжди мають однакову конфігурацію.



Підсистема мережі Kubernetes. Сервіси

621

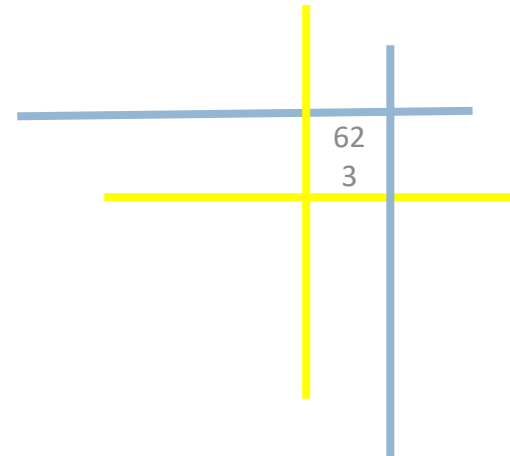
- ❑ Сервіси - це спеціальні об'єкти Kubernetes, які забезпечують доступ до контейнерів у кластері. Сервіси створюють DNS-імена та екземпляри IP-адрес, які можна використовувати для доступу до контейнерів.
- ❑ Сервіси забезпечують кілька переваг, включаючи:
 - Іменування контейнерів
 - Балансування навантаження
 - Розширення масштабу





Підсистема мережі Kubernetes. Політики мережі

- ❑ Політики мережі - це правила, які контролюють, як контейнери можуть спілкуватися один з одним. Політики мережі можна використовувати для забезпечення безпеки та контролю доступу до мережі.
- ❑ Політики мережі можна використовувати для виконання таких завдань, як:
 - Обмеження доступу до мережі
 - Забезпечення безпеки



Ресурси в Kubernetes



Сумісність Kubernetes та Докер₁

- ❑ Kubernetes підтримує контейнери Docker через програмний інтерфейс контейнерів (CRI). CRI - це стандартний інтерфейс, який дозволяє Kubernetes взаємодіяти з різними програмами для створення та управління контейнерами.
- ❑ Kubernetes поставляється з програмою CRI-Containerd, яка реалізує CRI. CRI-Containerd використовує Docker для створення та управління контейнерами.
- ❑ Це означає, що можна використовувати Docker для створення контейнерів, які можна використовувати в кластері Kubernetes. Не потрібно використовувати інший інструмент для створення контейнерів.



Сумісність Kubernetes та Докер₂

- ❑ Kubernetes також підтримує інші формати контейнерів, такі як CRI-O та rkt. Однак ці формати контейнерів не настільки поширені, як Docker.
- ❑ CRI-O - це програма CRI, яка не використовує Docker для створення та управління контейнерами. CRI-O може використовуватися для створення контейнерів з будь-якого формату контейнерів, включаючи Docker, CRI-O та rkt.
- ❑ rkt - це програма CRI, яка використовує власний формат контейнерів. rkt може використовуватися для створення контейнерів з формату контейнерів rkt.



Керування ресурсами у Kubernetes

- ❑ Kubernetes дозволяє керувати ресурсами, які виділяються контейнерам. Це можна зробити за допомогою обмежень і запитів ресурсів.
 - Ліміт - це максимальна кількість ресурсів, яку може використовувати контейнер. Якщо контейнер перевищує свій ліміт, він може бути зупинений Kubernetes.
 - Запит - це мінімальна кількість ресурсів, яку контейнер потребує для роботи. Якщо контейнер не може отримати достатньо ресурсів, він може працювати нестабільно або зупинитися.



Сховища у Kubernetes. Volumes

- ☐ Volumes - це об'єкти Kubernetes, які забезпечують доступ до даних для контейнерів. Volumes можуть бути локальними або зовнішніми.
- ☐ Локальні Volumes зберігаються на вузлі, на якому запускається контейнер. Локальні Volumes можуть бути корисні для зберігання тимчасових даних, таких як файли журналів або кеші.
- ☐ Зовнішні Volumes зберігаються на зовнішніх сховищах, таких як файлові системи, мережеві диски або хмарні сховища. Зовнішні Volumes можуть бути корисні для зберігання постійних даних, таких як база даних або файли користувачів.



Сховища у Kubernetes. Persistent Volumes

- ❑ Persistent Volumes - це спеціальний тип Volumes, який призначений для зберігання постійних даних. Persistent Volumes забезпечують стійкість даних, навіть якщо вузол, на якому вони розташовані, виходить з ладу.
- ❑ Persistent Volumes можна використовувати для зберігання таких даних, як база даних, файли користувачів або веб-сайти.



Огляд інструментів для розгортання

- ❑ Kubernetes надає вбудований інструмент командного рядка, `kubectl`, який можна використовувати для розгортання та управління застосунками. `kubectl` - це потужний інструмент, який може виконувати широкий спектр завдань, пов'язаних з Kubernetes.
- ❑ Однак `kubectl` може бути складним для використання, особливо для початківців. Існує також ряд інструментів сторонніх розробників, які можна використовувати для спрощення процесу розгортання.
- ❑ Деякі популярні інструменти сторонніх розробників для розгортання Kubernetes включають:
 - Helm - це пакетний менеджер, який використовується для розгортання та управління застосунками у Kubernetes. Helm робить процес розгортання більш простим і організованим.
 - FluxCD - це система оркестрування, яка використовується для автоматизації процесу розгортання. FluxCD може використовуватися для розгортання застосунку з репозиторія Git або з іншого джерела.
 - ArgoCD - це ще одна система оркестрування, яка використовується для автоматизації процесу розгортання. ArgoCD може використовуватися для розгортання застосунку з репозиторія Git або з іншого джерела.



Helm – пакетний менеджер Kubernetes.

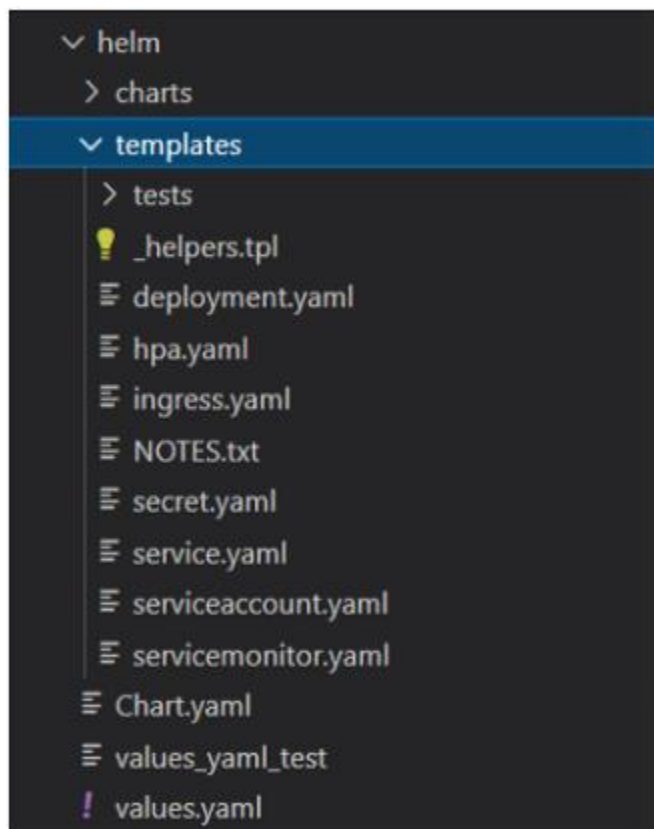
Структура Helm-чартів

- ❑ Helm-чарти - це архітектурні шаблони, які використовуються для розгортання, масштабування та керування додатками у Kubernetes. Чарти Helm містять всю інформацію, необхідну для розгортання додатка, включаючи визначення робочих процесів, служб, томів та інших ресурсів Kubernetes.
- ❑ Структура Helm-чартів:
 - `chart.yaml` - це файл метаданих, який містить загальну інформацію про чарт, таку як назва, версія та опис.
 - `values.yaml` - це файл конфігурації, який дозволяє користувачам налаштувати чарт відповідно до своїх потреб.
 - `templates` - це теки, які містять шаблони, які використовуються для генерації маніфестів Kubernetes.



Helm – пакетний менеджер Kubernetes.

chart.yaml





Helm – пакетний менеджер Kubernetes.

Установка та управління додатками

- ❑ Helm використовується для установки та управління додатками у Kubernetes за допомогою команди `helm install`. Ця команда приймає назву чарта як аргумент і розгортає чарт у кластері Kubernetes.
- ❑ Наприклад, наступна команда розгорне чарт із назвою "my-chart" у кластері Kubernetes:

```
helm install my-chart
```
- ❑ Для управління застосунком, розгорнутим за допомогою Helm, можна використовувати команди `helm upgrade` та `helm delete`.
 - Команда `helm upgrade` оновлює чарт, який вже розгорнутий у кластері Kubernetes.
 - Команда `helm delete` видаляє чарт із кластера Kubernetes.

Донецький національний технічний університет



Тема 8. Системи оркестрації.

Лекція 14. Конфігурація Kubernetes

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

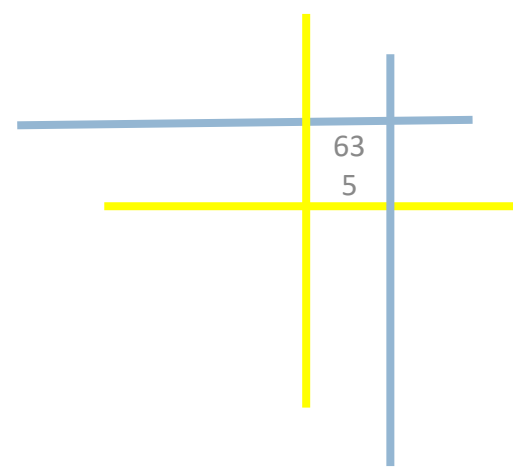
Укладач: проф. Дорогий Я.Ю.



Лекційні питання

634

- ☐ Конфігураційні файли Kubernetes
- ☐ Інші можливості Kubernetes



Конфігураційні файли Kubernetes



Структура YAML-файлів конфігурації

- ❑ YAML-файли конфігурації Kubernetes використовують синтаксис YAML для опису об'єктів Kubernetes. YAML - це формат текстового файлу, який використовується для описування структурованих даних.
- ❑ YAML-файли конфігурації Kubernetes поділяються на три основні частини:
 - *Версія* - це версія Kubernetes, для якої призначений файл конфігурації.
 - *Об'єкти* - це об'єкти Kubernetes, які описуються у файлі конфігурації.
 - *Конфігураційні параметри* - це параметри, які використовуються для налаштування.



Основні параметри та налаштування

- ❑ YAML-файли конфігурації Kubernetes містять ряд основних параметрів та налаштувань, які використовуються для опису об'єктів Kubernetes.
- ❑ Ось деякі з найбільш поширених параметрів та налаштувань:
 - Ім'я
 - Версія
 - Образ
 - Контейнери
 - Властивості



Приклад YAML-файлу конфігурації

□ Приклад YAML-файлу конфігурації, який описує робочий процес:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: my-image
```



Команди YAML файлу Kubernetes

- ❑ Kubernetes підтримує широкий спектр команд, які можна використовувати для створення, оновлення та видалення об'єктів Kubernetes. Ці команди використовуються для управління кластером Kubernetes.
- ❑ Ось деякі з основних команд Kubernetes:
 - `kubectl apply` - ця команда використовується для створення, оновлення або видалення об'єктів Kubernetes.
 - `kubectl get` - ця команда використовується для отримання інформації про об'єкти Kubernetes.
 - `kubectl describe` - ця команда використовується для отримання докладної інформації про об'єкти Kubernetes.
 - `kubectl edit` - ця команда використовується для редагування об'єктів Kubernetes.
 - `kubectl delete` - ця команда використовується для видалення об'єктів Kubernetes.



Використання `kubectl apply` для створення ресурсів

- ❑ `kubectl apply` - це команда, яка використовується для створення, оновлення або видалення об'єктів Kubernetes. Команда `kubectl apply` приймає як аргумент маніфест Kubernetes, який описує об'єкти, які потрібно створити.
- ❑ Наприклад, щоб створити робочий процес з трьома копіями, можна використовувати наступну команду:
 - `kubectl apply -f my-deployment.yaml`
 - Ця команда створить робочий процес з назвою "my-deployment" з трьома копіями.



Створення YAML-файлів за допомогою інструментів командного рядка

- ❑ Також можна створити YAML-файли за допомогою інструментів командного рядка, таких як `kubectl`.
- ❑ Щоб створити YAML-файл за допомогою `kubectl`, виконайте такі дії:
 - Виконайте команду `kubectl create`.
 - У параметрах команди вкажіть тип об'єкта, який потрібно створити.
 - У параметрах команди вкажіть параметри та налаштування об'єкта.
- ❑ Наприклад, щоб створити робочий процес з трьома копіями, ви можете виконати наступну команду:
 - `kubectl create deployment my-deployment --replicas=3`
 - Ця команда створить робочий процес з назвою "my-deployment" з трьома копіями.



Оновлення об'єктів

- ❑ Для оновлення об'єктів Kubernetes можна використовувати команду `kubectl edit`. Ця команда відкриває об'єкт у текстовому редакторі, де можна внести необхідні зміни.
- ❑ Наприклад, щоб оновити кількість копій робочого процесу, можна виконати наступну команду:
 - `kubectl edit deployment my-deployment`
 - Ця команда відкриє робочий процес "my-deployment" у текстовому редакторі. Можна змінити параметр `replicas` на бажане значення.



Видалення об'єктів

- ❑ Для видалення об'єктів Kubernetes можна використовувати команду `kubectl delete`. Ця команда приймає як аргумент ім'я об'єкта, який потрібно видалити.
- ❑ Наприклад, щоб видалити робочий процес, можна виконати наступну команду:
 - `kubectl delete deployment my-deployment`
 - Ця команда видалить робочий процес з назвою "my-deployment".



Отримання інформації про об'єкти

- ❑ Для отримання інформації про об'єкти Kubernetes можна використовувати команду `kubectl get`. Ця команда приймає як аргумент тип об'єкта та ім'я об'єкта.
- ❑ Наприклад, щоб отримати список всіх робочих процесів, можна виконати наступну команду:
 - `kubectl get deployments`
 - Ця команда видасть список всіх робочих процесів у кластері Kubernetes.



Додаткові команди

- ❑ Kubernetes підтримує ряд інших команд, які можна використовувати для виконання конкретних завдань.
- ❑ Наприклад, можна використовувати команду `kubectl logs` для перегляду журналів контейнерів, команду `kubectl port-forward` для перенаправлення порту контейнера на локальний порт, команду `kubectl scale` для масштабування об'єктів і так далі.



Довідка по ключових словах

- ❑ Довідку по ключових словах для YAML-файлів конфігурації Kubernetes можна отримати за допомогою команди `kubectl explain`. Ця команда виведе опис об'єкта Kubernetes і його ключових слів.



Ключові слова для YAML файлу Kubernetes

□ YAML-файли конфігурації Kubernetes використовують ряд ключових слів для опису об'єктів Kubernetes. Ось деякі з найпоширеніших ключових слів:

- `apiVersion` - це ключове слово, яке вказує версію Kubernetes, для якої призначений YAML-файл.
- `kind` - це ключове слово, яке вказує тип об'єкта Kubernetes.
- `metadata` - це ключове слово, яке описує метадані об'єкта Kubernetes.
- `spec` - це ключове слово, яке описує специфікацію об'єкта Kubernetes.



Інші ключові слова

❑ Деякі інші ключові слова, які використовуються в YAML-файлах конфігурації Kubernetes:

- `replicas` - це ключове слово, яке визначає кількість копій об'єкта.
- `selector` - це ключове слово, яке використовується для зв'язування об'єктів.
- `template` - це ключове слово, яке описує шаблон об'єкта.
- `ports` - це ключове слово, яке описує порти об'єкта.



Приклади використання ключових слів₁

- ☐ У цьому прикладі ключове слово `apiVersion` вказує версію Kubernetes, для якої призначений YAML-файл.
- ☐ Ключове слово `kind` вказує, що об'єкт є робочим процесом.
- ☐ Ключове слово `metadata` описує метадані робочого процесу, такі як ім'я та лейбли.
- ☐ Ключове слово `spec` описує специфікацію робочого процесу, зокрема кількість копій (3) та шаблон (один контейнер з образом Docker з назвою "my-image").

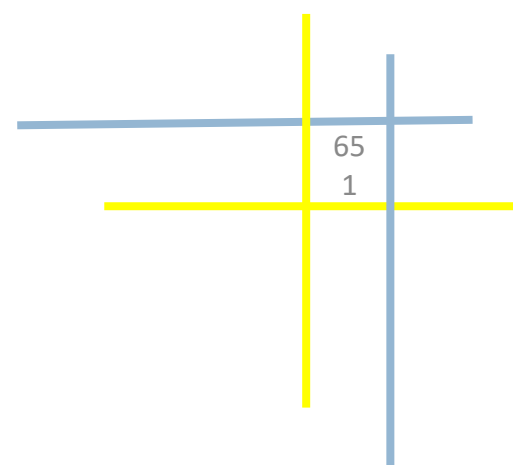
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: my-image
```



Приклади використання ключових слів₂

- ❑ У цьому прикладі ключове слово `apiVersion` вказує версію Kubernetes, для якої призначений YAML-файл.
- ❑ Ключове слово `kind` вказує, що об'єкт є службою.
- ❑ Ключове слово `metadata` описує метадані служби, такі як ім'я.
- ❑ Ключове слово `spec` описує специфікацію служби, зокрема лейбли, які використовуються для зв'язування з робочим процесом, та порти, які надаються службою.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: my-app
  ports:
    - port: 80
      targetPort: 8080
```



Інші можливості Kubernetes



Логи для відстеження подій

- ❑ Логи контейнерів містять інформацію про події, які відбуваються в контейнерах. Ці логи можна використовувати для відстеження стану розгортання.
- ❑ Щоб переглянути логи контейнерів, можна використовувати команду `kubectl logs`. Наприклад, щоб переглянути логи контейнера з ім'ям `"my-container"` у робочому процесі з ім'ям `"my-deployment"`, можна виконати наступну команду:
 - `kubectl logs deployments/my-deployment my-container`
 - Ця команда виведе логи контейнера `"my-container"` у робочому процесі `"my-deployment"`.



Інтеграція з системами моніторингу

- ❑ Kubernetes дозволяє інтегруватися з існуючими системами моніторингу, такими як Prometheus. Prometheus - це відкрита система моніторингу, яка використовує метри та логи для відстеження стану систем.
- ❑ Для інтеграції з Prometheus потрібно створити спеціальний об'єкт Kubernetes, який називається "PrometheusRule". Цей об'єкт описує, які параметри та логи потрібно збирати.



Централізований збір логів

- ❑ Kubernetes дозволяє централізовано збирати логи з контейнерів та системи. Для цього можна використовувати інструменти, такі як ELK Stack.
- ❑ ELK Stack - це відкрите рішення для логування, яке складається з Elasticsearch, Logstash та Kibana.
 - Elasticsearch - це інструмент для зберігання та пошуку логів.
 - Logstash - це інструмент для обробки логів.
 - Kibana - це інструмент для відображення логів.
- ❑ Для централізованого збору логів потрібно створити спеціальний об'єкт Kubernetes, який називається "ConfigMap". Цей об'єкт містить конфігурацію для інструментів логування.



Інтеграція Kubernetes з CI/CD. Створення пайплайнів CI/CD₁

- ❑ Пайплайни CI/CD (Continuous Integration/Continuous Delivery) - це автоматизовані процеси, які використовуються для розробки, тестування та розгортання програмного забезпечення.
- ❑ Для інтеграції Kubernetes з CI/CD потрібно створити пайплайн, який буде виконувати наступні кроки:
 - *Побудова та тестування програмного забезпечення.* У цьому кроці програмне забезпечення будується та тестується на локальній машині або в хмарі.
 - *Перевірка готовності до розгортання.* У цьому кроці перевіряється, чи готове програмне забезпечення до розгортання в Kubernetes.
 - *Розгортання програмного забезпечення в Kubernetes.* У цьому кроці програмне забезпечення розгортається в Kubernetes.



Інтеграція Kubernetes з CI/CD.

Створення пайплайнів CI/CD₂

- Для створення пайплайнів CI/CD можна використовувати різні інструменти, такі як Jenkins або GitLab CI.
- Jenkins - це відкрите програмне забезпечення для автоматизації завдань, яке можна використовувати для створення пайплайнів CI/CD.
 - GitLab CI - це система CI/CD, яка вбудована в GitLab.

```
pipeline {
  agent any

  stages {
    stage('Build') {
      steps {
        // Будує програмне забезпечення
      }
    }
    stage('Test') {
      steps {
        // Тестує програмне забезпечення
      }
    }
    stage('Deploy') {
      steps {
        // Розгортає програмне забезпечення в Kubernetes
      }
    }
  }
}
```



Використання Role-Based Access Control (RBAC)

- ❑ Role-Based Access Control (RBAC) - це система управління доступом, яка використовується для обмеження доступу до ресурсів Kubernetes. RBAC дозволяє надавати користувачам або групам користувачів певні ролі, які визначають, які дії вони можуть виконувати.
- ❑ Для використання RBAC потрібно створити ролі та надати їх користувачам або групам користувачів. Ролі можна створювати за допомогою команди `kubectl create role`.
- ❑ Ось приклад ролі, яка дозволяє користувачам створювати, оновлювати та видаляти робочі процеси:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: my-role
rules:
- apiGroups:
  - apps
  resources:
  - deployments
  verbs:
  - create
  - update
  - delete
```



Налаштування мережевих політик для обмеження доступу

- ❑ Kubernetes дозволяє налаштовувати мережеві політики для обмеження доступу до ресурсів. Мережеві політики можна використовувати для блокування доступу з певних джерел або для забезпечення доступу тільки до певних ресурсів.
- ❑ Для налаштування мережевих політик потрібно створити мережеві правила. Мережеві правила можна створювати за допомогою команди `kubectl create networkpolicy`.
- ❑ Ось приклад мережевого правила, яке блокує доступ з зовнішньої мережі до всіх робочих процесів:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-networkpolicy
spec:
  podSelector:
    matchLabels:
      app: my-app
  ingress: []
  egress:
    - to:
      - podSelector:
          matchLabels:
            app: my-app
```



Приклади інструментів для забезпечення безпеки

❑ Ось кілька прикладів інструментів для забезпечення безпеки, які можна використовувати з Kubernetes:

- OPA (Open Policy Agent) - це інструмент для управління політиками, який можна використовувати для реалізації політики безпеки в Kubernetes.
- Istio - це інструмент для мікросервісної архітектури, який включає в себе функції безпеки, такі як шифрування та аутентифікація.
- Cilium - це інструмент для мережевої безпеки, який можна використовувати для реалізації політики безпеки в Kubernetes на рівні мережі.

❑ Вибір інструментів залежить від конкретних потреб та бюджету.



Розгортання нових версій застосунку за допомогою Rollouts

- ❑ Rollouts - це механізм, який дозволяє розгортати нові версії застосунків у Kubernetes поступово, щоб мінімізувати ризик збоїв.
- ❑ Rollouts працюють шляхом створення нового робочого процесу з новою версією застосунку та поступового перемикавання трафіка з старого робочого процесу на новий.
- ❑ Існує кілька типів rollouts:
 - `RollingUpdate` - це базовий тип `rollout`, який перемикає весь трафік на новий робочий процес одночасно.
 - `Recreate` - це тип `rollout`, який видаляє старий робочий процес і створює новий.
 - `Canary` - це тип `rollout`, який перемикає трафік на новий робочий процес поступово, починаючи з невеликої кількості користувачів.



Стратегії масштабування для ефективного використання ресурсів

- ❑ Kubernetes дозволяє масштабувати додатки за допомогою різних стратегій.
- ❑ Однією з найпоширеніших стратегій масштабування є `HorizontalPodAutoscaler` (HPA). HPA - це об'єкт Kubernetes, який автоматично масштабує робочий процес відповідно до навантаження.
- ❑ HPA працює шляхом моніторингу метрик, таких як завантаження CPU або використання пам'яті. Коли навантаження перевищує певний рівень, HPA автоматично збільшує кількість копій робочого процесу.
- ❑ Інші стратегії масштабування включають:
 - `Manual scaling` - це ручне масштабування, яке виконується користувачем.
 - `Vertical scaling` - це масштабування, яке збільшує розмір контейнерів у робочому процесі.
 - `Autoscaling based on resource utilization` - це масштабування, яке збільшує або зменшує кількість копій робочого процесу відповідно до використання ресурсів.
 - `Autoscaling based on custom metrics` - це масштабування, яке збільшує або зменшує кількість копій робочого процесу відповідно до власних метрик.
- ❑ Вибір стратегії масштабування залежить від конкретних потреб застосунку.



Використання резервування та реплікації

- ❑ Резервування та реплікація - це два важливих механізми, які можна використовувати для забезпечення стійкості системи.
- ❑ *Резервування* - це створення копії ресурсу, яку можна використовувати в разі несправності основного ресурсу. Наприклад, для резервування контейнера можна використовувати об'єкт Kubernetes "ReplicaSet".
- ❑ *Реплікація* - це створення декількох копій ресурсу, щоб забезпечити його доступність навіть у разі несправності частини копій. Наприклад, для реплікації контейнера можна використовувати об'єкт Kubernetes "Deployment".



Обробка відмов та автоматичне відновлення

- ❑ Обробка відмов - це процес, який дозволяє системі відновитися після несправності. Автоматичне відновлення - це процес, який дозволяє системі відновитися без втручання людини.
- ❑ Для обробки відмов та автоматичного відновлення можна використовувати різні інструменти та практики. Наприклад, можна використовувати такі інструменти, як Kubernetes HorizontalPodAutoscaler (HPA) та Kubernetes Ingress Controller.

Донецький національний технічний університет



Тема 9. Системи моніторингу та налаштування сервісів.

Лекція 15. Система моніторингу Prometheus

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

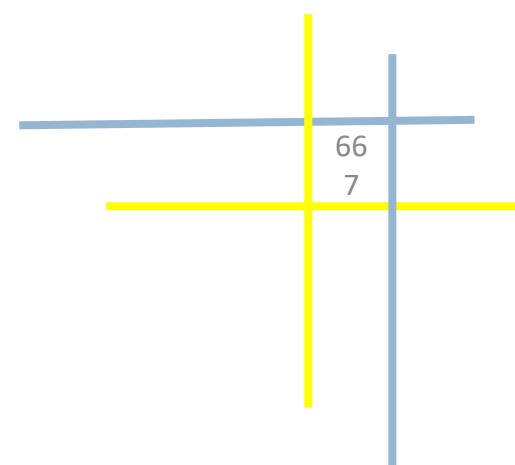
665

- ☐ Вступ
- ☐ Архітектура Prometheus
- ☐ PromQL
- ☐ Встановлення та налаштування Prometheus
- ☐ Моніторинг та візуалізація даних
- ☐ Безпека Prometheus



Огляд Prometheus

- ❑ Prometheus - це відкрите програмне забезпечення для моніторингу, яке використовується для збирання, зберігання та візуалізації метрик.
- ❑ Prometheus використовує модель push-моніторингу, в якій агенти моніторингу (exporters) збирають метрики з об'єктів, які потрібно моніторити, і надсилають їх до сервера Prometheus.
- ❑ Prometheus зберігає метрики в базі даних часу серії (time series database), яка дозволяє збирати та зберігати дані протягом тривалого періоду часу.
- ❑ Prometheus також надає потужну систему візуалізації, яка дозволяє створювати різні графіки та діаграми для аналізу даних моніторингу.



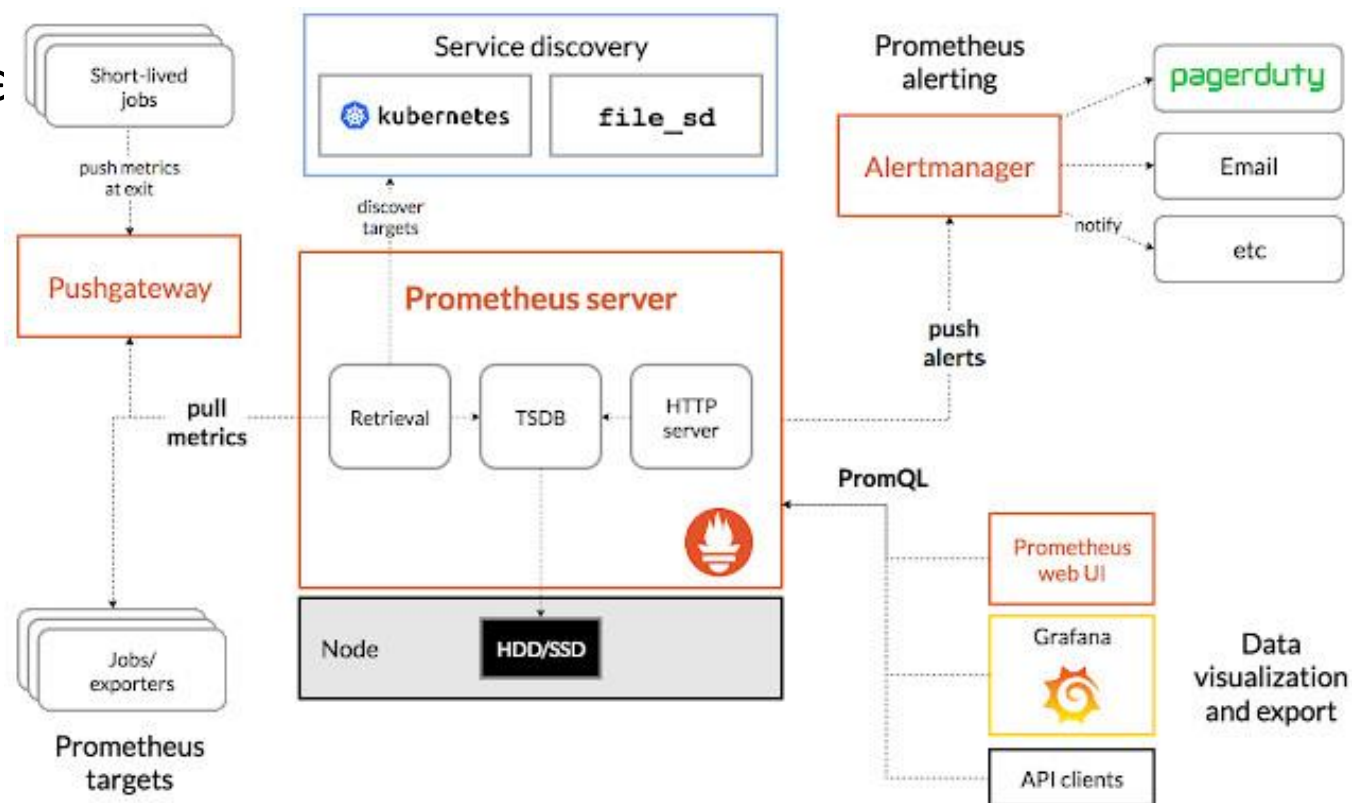
Архітектура Prometheus



Архітектура Prometheus₁

❑ Основні компоненти:

- Сервер Prometheus - це основний компонент Prometheus. Він відповідає за збирання, зберігання та візуалізацію метрик.
- Збирачі - це агенти моніторингу, які збирають метрики з об'єктів, які потрібно моніторити.
- Зберігачі даних - це бази даних часу серії, які використовуються для зберігання метрик.





Архітектура Prometheus₂

□ Prometheus - модульна моніторингова система, її підсистеми за потреби можуть бути замінені на альтернативні. Проте основні елементи Prometheus наступні:

- `Prometheus server` (Web UI/API, TSDB) - основний сервер моніторингу, запитує та зберігає всі метрики від клієнтів, надає базову веб-панель запитів та API
- `Exporters` - компоненти, що виступають джерелом даних для Prometheus сервера. Зазвичай працюють на стороні клієнта, але є і винятки, наприклад, `Blackbox exporter`. Є купа готових як офіційних, так і сторонніх рішень для збирання метрик з MySQL, RabbitMQ та ін.
- `Alertmanager` - підсистема, що забезпечує групування, стримування, дедуплікацію, маршрутизацію повідомлень від систем на зразок серверу Prometheus. Також має базову веб-панель, де можна відфільтрувати повідомлення, призупинити їх надсилання при певних умовах та ін.
- `Pushgateway` - служба-посередник, що на відміну від Prometheus server, отримує метрики від вузлів і вже потім пересилає їх на основний сервер. Не обов'язковий компонент, але може бути корисний у разі моніторингу сервісів із коротким життєвим циклом чи програм, що не надають відкритий порт для отримання метрик основним сервером.
- `Grafana` - веб-панель на Go/NodeJS для побудови та відображення метрик. Вміє працювати з Prometheus "із коробки".



Взаємодія між компонентами

- ❑ Збирачі збирають метрики з об'єктів, які потрібно моніторити. Вони можуть збирати метрики з різних джерел, таких як:
 - Контейнери
 - Сервери
 - Мережеві пристрої
 - Бази даних
- ❑ Зібране збирачем метрики надсилаються до сервера Prometheus. Сервер Prometheus зберігає метрики в базі даних часу серії.
- ❑ Для візуалізації метрик можна використовувати Prometheus UI або сторонні інструменти, такі як Grafana.

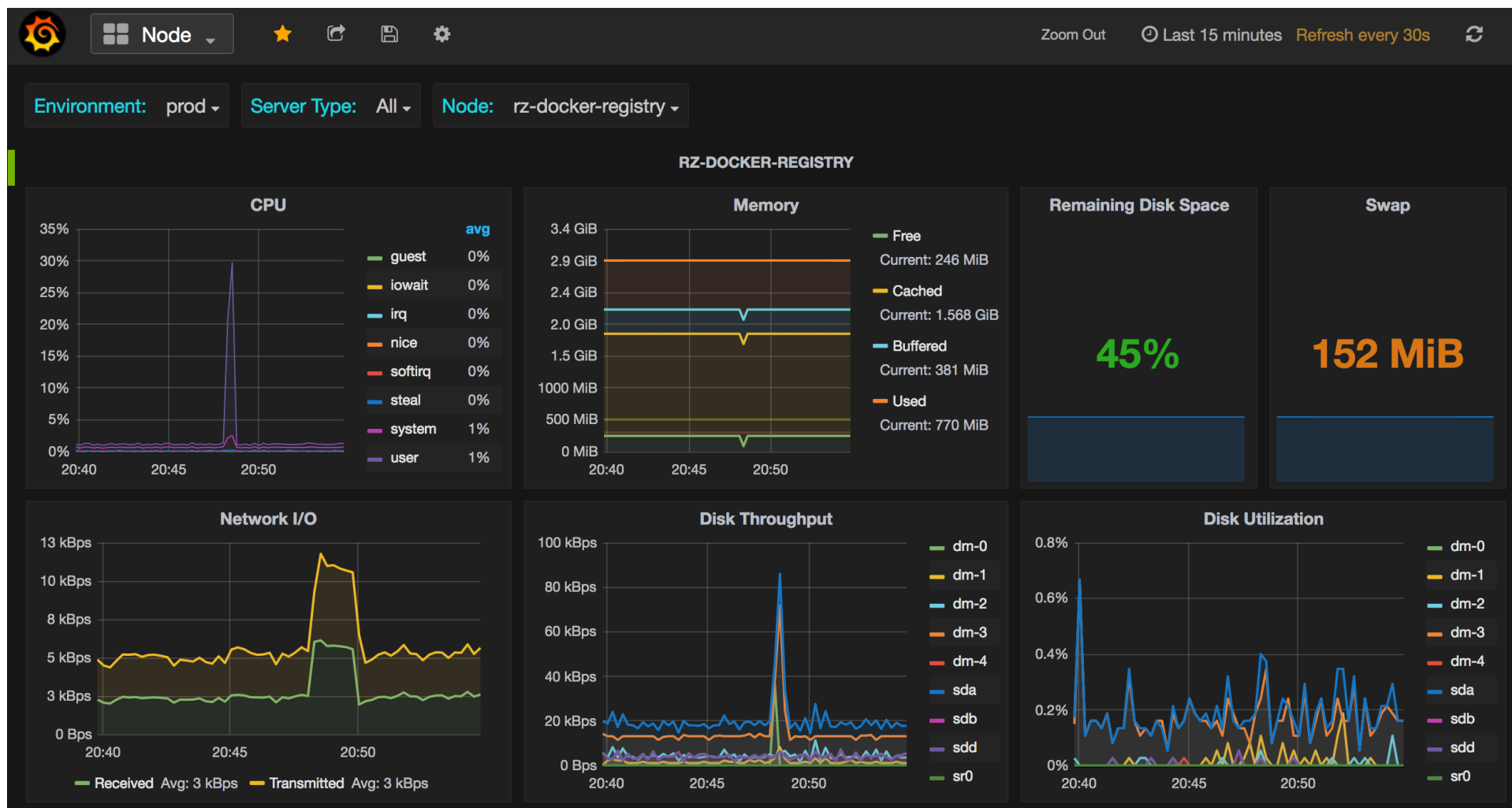


Інтерфейс користувача Prometheus₁

- ☐ Веб-інтерфейс Prometheus (UI) - це простий і інтуїтивно зрозумілий інструмент для візуалізації метрик.
- ☐ UI дозволяє створювати різні графіки та діаграми, щоб аналізувати дані моніторингу.
- ☐ UI використовує мову запитів PromQL для запитування метрик.
- ☐ PromQL - це потужна мова, яка дозволяє створювати складні запити для аналізу даних.
- ☐ Також UI дозволяє фільтрувати дані за допомогою різних критеріїв, таких як:
 - Мітка
 - Вартість
 - Період часу



Інтерфейс користувача Prometheus₂





Збирачі даних (Exporters)

- ❑ Збирачі даних (exporters) - це агенти моніторингу, які збирають метрики з об'єктів, які потрібно моніторити. Зібране збирачем метрики надсилаються до сервера Prometheus.
- ❑ Prometheus використовує збирачів для моніторингу різноманітних систем, таких як:
 - Контейнери
 - Сервери
 - Мережеві пристрої
 - Бази даних
 - Файлові системи
 - Веб-сервери



Приклади експортерів

□ Деякі приклади збирачів даних:

- *Node Exporter* - це збирач, який збирає метрики з серверів, таких як використання CPU, використання пам'яті, використання диску та мережний трафік.
- *MySQL Exporter* - це збирач, який збирає метрики з баз даних MySQL, такі як використання CPU, використання пам'яті, кількість відкритих з'єднань та кількість запитів.
- *Prometheus Operator* - це збирач, який використовується для управління збирачами даних в Kubernetes.



Автоматизація конфігурації збирачів

- ❑ Сервісні дискавері-системи - це системи, які динамічно виявляють ресурси в мережі. Вони можуть використовуватися для автоматизації конфігурації збирачів Prometheus.
- ❑ Наприклад, Prometheus можна налаштувати для використання сервісної дискавері-системи для виявлення контейнерів в Kubernetes. У цьому випадку Prometheus буде автоматично збирати метрики з усіх контейнерів, які виявлені сервісною дискавері-системою.



Використання Alertmanager

- ❑ Alertmanager - це інструмент, який використовується для обробки і надсилання сповіщень про тривоги, які генеруються Prometheus. Alertmanager дозволяє налаштовувати сповіщення за допомогою шаблонів, які дозволяють гнучко визначати, як і коли надсилати сповіщення.
- ❑ Alertmanager може надсилати сповіщення різними способами, наприклад, за допомогою електронної пошти, SMS, Slack або webhooks.



Масштабування Prometheus

- ❑ Федерация - це спосіб об'єднання декількох серверів Prometheus в єдину систему моніторингу. Федерация дозволяє масштабувати Prometheus, розподіляючи навантаження між декількома серверами.
- ❑ Для використання федерації потрібно налаштувати два або більше серверів Prometheus. Сервери Prometheus, які беруть участь у федерації, називаються федеративними серверами.
- ❑ Федерация працює шляхом обміну метриками між федеративними серверами. Це дозволяє збирати метрики з більшої кількості систем, ніж може бути доступно з одного сервера Prometheus.



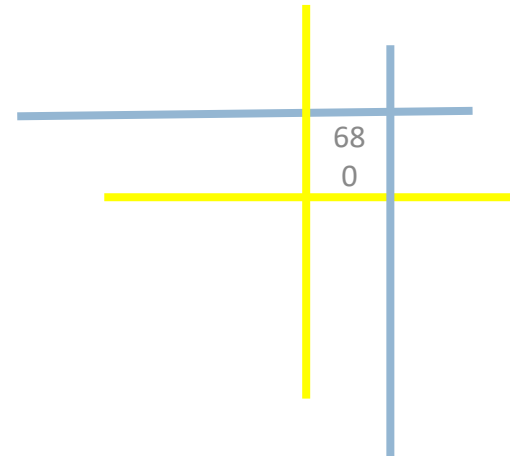
Горизонтальне масштабування

- ❑ Горизонтальне масштабування - це спосіб масштабування Prometheus, додаючи додаткові сервери Prometheus. Горизонтальне масштабування дозволяє збільшувати пропускну здатність та продуктивність Prometheus.
- ❑ Для горизонтального масштабування потрібно налаштувати кілька серверів Prometheus. Сервери Prometheus, які додаються до системи, називаються додатковими серверами.
- ❑ Додаткові сервери Prometheus додаються до кластера Prometheus за допомогою конфігураційного файлу.



Федерація Prometheus

- ❑ Федерация Prometheus - це механізм, який дозволяє об'єднати різні інстанції Prometheus в єдину систему. Це дозволяє збирати дані з різних систем і середовищ в одному місці.
- ❑ Для використання федерації Prometheus потрібно налаштувати кожну інстанцію Prometheus для відправки даних в центральний сервер федерації. Центральний сервер федерації збирає дані з усіх інстанцій Prometheus і зберігає їх в єдиному місці.



PromQL



Запити

❑ Запити використовуються для отримання даних з баз даних Prometheus. Вони пишуться мовою PromQL.

❑ Простий приклад запиту:

```
# Get the CPU usage for the last 5 minutes  
rate(container_cpu_usage_seconds_total[5m])
```

- Цей запит поверне значення використання CPU для всіх контейнерів за останні 5 хвилин.



Фільтрація₁

□ Фільтрація використовується для обмеження даних, які повертаються запитом. Вона використовується для таких завдань, як:

- Обмеження даних за мітками
- Обмеження даних за періодом часу
- Обмеження даних за значенням

□ Приклад фільтрації за мітками:

```
# Get the CPU usage for the containers with the label  
"app=nginx"  
rate(container_cpu_usage_seconds_total{app="nginx"}[5m])
```

- Цей запит поверне значення використання CPU для всіх контейнерів з міткою "app=nginx" за останні 5 хвилин.



Фільтрація₁

□ Фільтрація використовується для обмеження даних, які повертаються запитом. Вона використовується для таких завдань, як:

- Обмеження даних за мітками
- Обмеження даних за періодом часу
- Обмеження даних за значенням

□ Приклад фільтрації за мітками:

```
# Get the CPU usage for the containers with the label  
"app=nginx"  
rate(container_cpu_usage_seconds_total{app="nginx"}[5m])
```

- Цей запит поверне значення використання CPU для всіх контейнерів з міткою "app=nginx" за останні 5 хвилин.



Фільтрація₂

□ Приклад фільтрації за періодом часу:

```
# Get the CPU usage for the last 1 hour  
rate(container_cpu_usage_seconds_total[1h])
```

- Цей запит поверне значення використання CPU для всіх контейнерів за останні 1 годину.

□ Приклад фільтрації за значенням:

```
# Get the CPU usage for the containers with CPU usage  
greater than 50%  
rate(container_cpu_usage_seconds_total[5m]) > 0.5
```

- Цей запит поверне значення використання CPU для всіх контейнерів, у яких використання CPU перевищує 50% за останні 5 хвилин.



Основні концепції мови запитів Prometheus

- ❑ Мова запитів Prometheus (PromQL) - це функціональна мова запитів, який використовується для запитування метрик з баз даних Prometheus.
- ❑ PromQL має наступні основні концепції:
 - Серія - це послідовність даних, що представляє значення метрики протягом певного періоду часу.
 - Мітка - це додаткова інформація, яка додається до даних для їх ідентифікації.
 - Функція - це функція, яка використовується для обробки даних.



Приклади запитів для отримання даних₁

❑ Отримати значення метрики

```
# Get the CPU usage for the last 5 minutes  
rate(container_cpu_usage_seconds_total[5m])
```

- Цей запит поверне значення використання CPU для всіх контейнерів за останні 5 хвилин.

❑ Отримати середнє значення метрики

```
# Get the average CPU usage for the last 5 minutes  
avg(rate(container_cpu_usage_seconds_total[5m]))
```

- Цей запит поверне середнє значення використання CPU для всіх контейнерів за останні 5 хвилин.

❑ Отримати максимальне значення метрики

```
# Get the maximum CPU usage for the last 5 minutes  
max(rate(container_cpu_usage_seconds_total[5m]))
```

- Цей запит поверне максимальне значення використання CPU для всіх контейнерів за останні 5 хвилин.



Приклади запитів для отримання даних₂

❑ Отримати значення метрики за мітками

```
# Get the CPU usage for the containers with the label "app=nginx"
rate(container_cpu_usage_seconds_total{app="nginx"}[5m])
```

- Цей запит поверне значення використання CPU для всіх контейнерів з міткою "app=nginx" за останні 5 хвилин.

❑ Отримати значення метрики за періодом часу

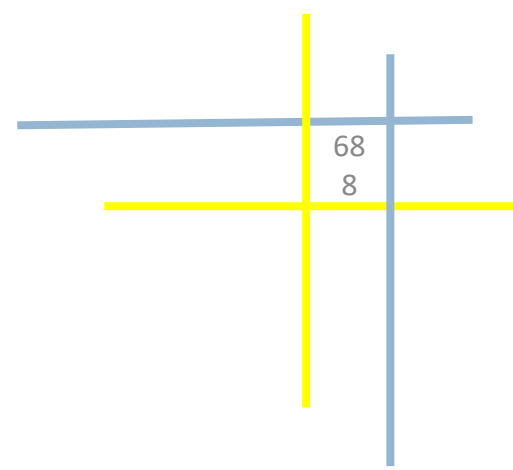
```
# Get the CPU usage for the last 1 hour
rate(container_cpu_usage_seconds_total[1h])
```

- Цей запит поверне значення використання CPU для всіх контейнерів за останні 1 годину.

❑ Отримати значення метрики за значенням

```
# Get the CPU usage for the containers with CPU usage greater than 50%
rate(container_cpu_usage_seconds_total[5m]) > 0.5
```

- Цей запит поверне значення використання CPU для всіх контейнерів, у яких використання CPU перевищує 50% за останні 5 хвилин.



Встановлення та налаштування Prometheus



Встановлення та запуск Prometheus

- ☐ Prometheus можна встановити різними способами, наприклад, за допомогою пакетного менеджера, за допомогою Docker або за допомогою збірника інструментів.
- ☐ Приклад встановлення Prometheus за допомогою пакетного менеджера в дистрибутиві Linux:
 - `sudo apt install prometheus`
- ☐ Після встановлення Prometheus можна запустити його за допомогою команди:
 - `prometheus`
- ☐ Prometheus за замовчуванням запускається на порту 9090. Щоб отримати доступ до Prometheus, потрібно відкрити веб-браузер і перейти за адресою `http://localhost:9090`.



Конфігураційні файли та їх основні параметри₁

- Prometheus використовує два конфігураційних файли:
- `prometheus.yml` - основний конфігураційний файл, який визначає основні параметри Prometheus, такі як порт, на якому він запускається, і список збирачів, які він використовує.
 - `alertmanager.yml` - конфігураційний файл для Alertmanager, який використовується для обробки сповіщень.



Конфігураційні файли та їх основні параметри₂

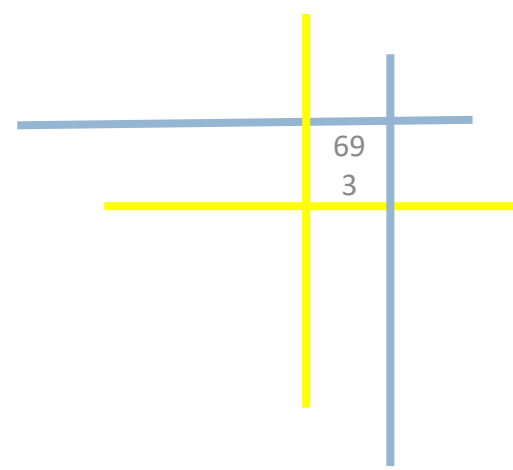
❑ Основні параметри, які можна налаштувати в конфігураційному файлі Prometheus:

- `listen_address`: адреса, на якій Prometheus слухає запити.
- `port`: порт, на якому Prometheus слухає запити.
- `scrape_interval`: інтервал, з яким Prometheus збирає метрики від збирачів.
- `scrape_configs`: список збирачів, які Prometheus використовує для збору метрик.



Динамічне виявлення нових ресурсів

- ❑ Prometheus може динамічно виявляти нові ресурси, які починають генерувати метрики. Це дозволяє Prometheus швидко реагувати на зміни в системі, яка моніториться.
- ❑ Наприклад, Prometheus можна налаштувати для динамічного виявлення нових контейнерів, які запускаються в Kubernetes. У цьому випадку Prometheus буде автоматично додавати збирачі для цих контейнерів, коли вони будуть запущені.



Моніторинг та візуалізація даних



Налаштування правил тривоги

- ❑ Тривоги - це сигнали, які генеруються Prometheus, коли значення метрики виходить за межі допустимого діапазону. Тривоги використовуються для інформування користувачів про потенційні проблеми в системах, які моніторяться.
- ❑ Правила тривоги використовуються для визначення того, коли тривоги повинні генеруватися. Правила тривоги містять наступну інформацію:
 - Метрика, за якою потрібно стежити.
 - Діапазон, за яким значення метрики виходить за межі допустимого.
 - Тип тривоги, яка повинна генеруватися.
 - Адреса, на яку необхідно надіслати сповіщення про тривогу.



Конфігурація зберігачів даних

- ❑ Prometheus використовує зберігачі даних для зберігання метрик, які він збирає. Зберігач даних - це програма, яка зберігає дані в довгостроковій пам'яті.
- ❑ Prometheus підтримує кілька типів зберігачів даних, таких як:
 - InfluxDB
 - Grafana Cloud
 - OpenTSDB
 - SQLite
- ❑ Для використання зберігача даних потрібно налаштувати його в конфігураційному файлі Prometheus.



Використання вбудованого засобу архівування даних

- ☐ Prometheus включає в себе вбудований засіб архівування даних, який дозволяє зберігати дані протягом тривалого періоду часу. Вбудований засіб архівування даних зберігає дані в форматі, який можна використовувати для відтворення графіків і діаграм.
- ☐ Для використання вбудованого засобу архівування даних потрібно налаштувати його в конфігураційному файлі Prometheus.



Використання графіків для аналізу даних

- ❑ Графіки - це ефективний спосіб візуалізації даних. Вони дозволяють швидко і легко зрозуміти тенденції, аномальні значення та інші важливі відомості про дані.
- ❑ Prometheus включає в себе вбудований веб-інтерфейс, який дозволяє створювати графіки на основі даних, які він збирає. Вбудований веб-інтерфейс дозволяє створювати прості графіки, але для більш складних графіків можна використовувати сторонні інструменти візуалізації.



Інтеграція Prometheus з іншими інструментами візуалізації

- ❑ Prometheus можна інтегрувати з іншими інструментами візуалізації, такими як Grafana, Kibana та ELK.
- ❑ Ця інтеграція дозволяє створювати складні графіки та діаграми, які можуть допомогти в аналізі даних.



Інтеграція Prometheus з Grafana

- ❑ Grafana - це інструмент візуалізації даних, який можна використовувати для створення графіків, діаграм та інших видів візуалізацій на основі даних, які збирає Prometheus.
- ❑ Інтеграція Prometheus з Grafana дозволяє створювати складні графіки та діаграми, які можуть допомогти в аналізі даних.



Встановлення та налаштування зв'язку між системами

- ☐ Для інтеграції Prometheus з Grafana потрібно встановити обидва інструменти.
- ☐ Після установки Prometheus і Grafana потрібно налаштувати зв'язок між ними.
- ☐ Для цього потрібно виконати такі кроки:
 - У конфігураційному файлі Prometheus потрібно вказати URL-адресу сервера Grafana.
 - У конфігураційному файлі Grafana потрібно вказати URL-адресу сервера Prometheus.
- ☐ Після налаштування зв'язку між системами можна почати створювати графіки та діаграми на основі даних, які збирає Prometheus.



Використання Prometheus для моніторингу контейнерів Docker₁

- ❑ Prometheus можна використовувати для моніторингу контейнерів Docker. Для цього потрібно встановити Prometheus і контейнерний експортер.
- ❑ Контейнерний експортер - це програма, яка збирає метрики з контейнера Docker і надсилає їх Prometheus.
- ❑ Prometheus підтримує кілька контейнерних експортерів, таких як:
 - *Docker exporter* - це офіційний контейнерний експортер Prometheus.
 - *cAdvisor* - це контейнерний експортер, який збирає широку гаму метрик, включаючи використання CPU, використання пам'яті, використання диску та мережний трафік.
 - *Prometheus exporter for Kubernetes* - це контейнерний експортер, який збирає метрики з контейнерів, які працюють в Kubernetes.



Використання Prometheus для моніторингу контейнерів Docker₂

- ❑ Для використання Prometheus для моніторингу контейнерів Docker потрібно виконати такі кроки:
 - Встановити Prometheus.
 - Встановити контейнерний експортер.
 - Налаштувати контейнерний експортер для надсилання метрик Prometheus.
- ❑ Після налаштування контейнерного експортера можна почати створювати графіки та діаграми на основі даних, які збирає Prometheus.



Використання Prometheus для моніторингу контейнерів Docker₃

- ❑ Контейнерні експортери можна використовувати для моніторингу різних аспектів контейнерів Docker. Деякі приклади того, що можна моніторити за допомогою контейнерних експортерів:
 - Використання ресурсів, таких як використання CPU, використання пам'яті та використання диску.
 - Продуктивність, такі як кількість запитів, час відгуку та помилки.
 - Безпека, такі як кількість вхідних підключень, кількість шкідливих атак та кількість помилок аутентифікації.
- ❑ Контейнерні експортери можуть використовуватися для моніторингу контейнерів Docker в будь-якому середовищі. Вони є простим і ефективним способом моніторингу контейнерів Docker.



Правила тревов

- ❑ Правила тривог - це важливий компонент системи моніторингу Prometheus. Вони дозволяють інформувати користувачів про потенційні проблеми в системах, які моніторяться.
- ❑ Правила тривог генеруються, коли значення метрики виходить за межі допустимого діапазону. Тривоги можуть бути генеровані для різних типів метрик, таких як використання CPU, використання пам'яті, використання диску та мережний трафік.
- ❑ Тривоги є важливим інструментом для оперативного реагування на проблеми. Вони дозволяють користувачам швидко дізнатися про проблеми і вжити заходів для їх усунення.



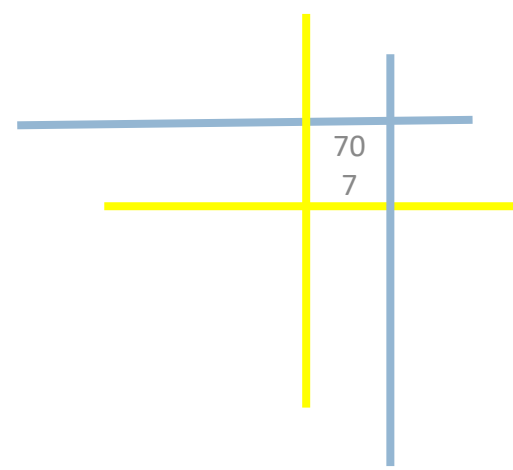
Робота з метриками для більш точного моніторингу

- ❑ Метрики - це основні одиниці даних, які збирає Prometheus. Вони представляють різні аспекти стану системи, такі як використання ресурсів, продуктивність та метрики безпеки.
- ❑ Для більш точного моніторингу важливо правильно працювати з метриками. Ось кілька порад:
 - Визначте, які метрики потрібно збирати. Не потрібно збирати всі доступні метрики. Збирайте тільки ті метрики, які важливі для моніторингу ваших систем.
 - Надавайте метрики іменами, які відображають їх значення. Це допоможе вам легко зрозуміти, що означає кожна метрика.
 - Використовуйте етикетки для класифікації даних. Етикетки дозволяють вам групувати дані разом на основі спільних характеристик.



Використання етикеток для класифікації даних

- ☐ Етикетки - це додаткові атрибути, які можна додати до метрик. Вони дозволяють вам класифікувати дані на основі спільних характеристик.
- ☐ Наприклад, ви можете використовувати етикетки для класифікації даних за такими параметрами:
 - Ім'я системи
 - Версія програмного забезпечення
 - Вік системи
 - Фізичне розташування системи
- ☐ Класифікація даних за допомогою етикеток може допомогти вам в аналізі даних і виявленні пробл



Безпека Prometheus



Захист Prometheus

- ❑ Для використання SSL/TLS потрібно створити сертифікати SSL/TLS для кожного компонента Prometheus. Ці сертифікати можна створити за допомогою сертифікаційного органу або самостійно.
- ❑ Після створення сертифікатів SSL/TLS потрібно налаштувати компоненти Prometheus для використання цих сертифікатів.



Ролева модель доступу та налаштування прав користувачів

- ☐ Prometheus підтримує ролеву модель доступу (RBAC), яка дозволяє налаштовувати права користувачів для доступу до даних.
- ☐ RBAC дозволяє надавати користувачам різні рівні доступу до даних, такі як:
 - Читання - можливість читати дані.
 - Запис - можливість записувати дані.
 - Адміністрування - можливість налаштовувати систему.
- ☐ Для налаштування RBAC потрібно створити ролі і надати користувачам права до цих ролей.



Моніторинг безпеки

- ❑ Prometheus можна використовувати для моніторингу безпеки системи. Для цього потрібно зібрати метрики безпеки з різних джерел, таких як:
 - *Сервери журналів* - можуть генерувати метрики про події безпеки, такі як спроби входу в систему, спроби доступу до заборонених ресурсів та інші.
 - *Системи аудиту* - можуть генерувати метрики про події безпеки, такі як зміни конфігурації, додавання нових користувачів та інші.
 - *Системи виявлення вторгнень (IDS/IPS)* - можуть генерувати метрики про потенційні загрози безпеки, такі як спроби атак, виявлення шкідливого програмного забезпечення та інші.
- ❑ Після збору метрик безпеки можна використовувати Prometheus для створення графіків і діаграм, які допоможуть в аналізі даних і виявленні проблем безпеки.



Спостереження за подіями

- ❑ Одним із способів використання Prometheus для моніторингу безпеки є спостереження за подіями. Події безпеки - це повідомлення, які генеруються системами безпеки, коли відбувається щось важливе, наприклад, спроба входу в систему або спроба доступу до забороненого ресурсу.
- ❑ Prometheus можна використовувати для моніторингу подій безпеки, налаштувавши правила тривоги. Правила тривоги генеруються, коли значення метрики виходить за межі допустимого діапазону.
- ❑ Наприклад, можна налаштувати правило тривоги, яке генерується, коли кількість спробувань входу в систему перевищує певний пороговий рівень. Це може допомогти виявити потенційні атак



Аудит

- ❑ Аудит - це процес перевірки системи на наявність уразливостей безпеки. Prometheus можна використовувати для моніторингу аудиту, налаштувавши правила тривоги для метрик аудиту.
- ❑ Наприклад, можна налаштувати правило тривоги, яке генерується, коли кількість змін конфігурації перевищує певний пороговий рівень. Це може допомогти виявити потенційні зміни безпеки.



Виявлення інцидентів

- ❑ Виявлення інцидентів - це процес виявлення і реагування на потенційні загрози безпеки. Prometheus можна використовувати для виявлення інцидентів, аналізуючи дані з різних джерел.
- ❑ Наприклад, можна використовувати Prometheus для виявлення аномалій у даних безпеки. Аномалії можуть бути ознакою потенційної загрози безпеки.



Оновлення та резервне копіювання

- ❑ Оновлення Prometheus - це важливий процес, який потрібно виконувати регулярно. Оновлення Prometheus може включати в себе оновлення версій програмного забезпечення, оновлення даних або оновлення конфігурацій.
- ❑ Для правильного оновлення Prometheus потрібно дотримуватися таких кроків:
 - *Плануйте оновлення.* Перед оновленням Prometheus потрібно спланувати процес оновлення. Це допоможе уникнути проблем під час оновлення.
 - *Зробіть резервну копію.* Перед оновленням Prometheus потрібно зробити резервну копію конфігурацій, правил тривоги та даних. Це допоможе відновити систему в разі проблем під час оновлення.
 - *Ознайомтеся з новими функціями.* Перед оновленням Prometheus потрібно ознайомитися з новими функціями. Це допоможе правильно налаштувати систему після оновлення.
 - *Оновіть Prometheus.* Після завершення планування, створення резервної копії та ознайомлення з новими функціями можна виконувати оновлення Prometheus.

Донецький національний технічний університет



Тема 9. Системи моніторингу та налаштування сервісів. Лекція 16. Система налаштування сервісів інфраструктури Consul.

Предмет: Технології DevOPS

Розділ: Кібербезпека

Галузь: Інформаційні технології

Укладач: проф. Дорогий Я.Ю.



Лекційні питання

716

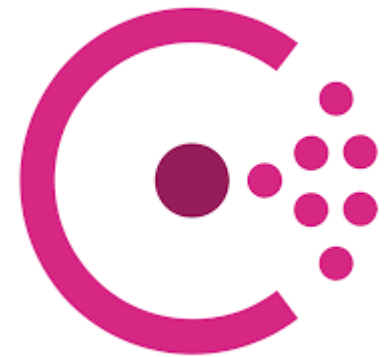
- ☐ Вступ
- ☐ Архітектура Consul
- ☐ Конфігурація Consul
- ☐ Безпека Consul
- ☐ Моніторинг Consul

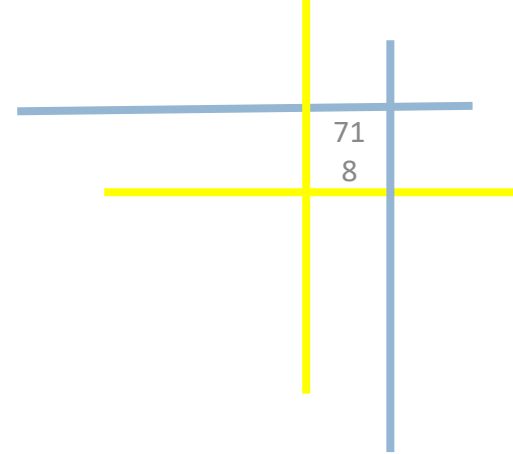


Ознайомлення із системою Consul

□ Consul - це відкрита система управління конфігурацією і служба реєстру ключів, яка використовується для керування розподіленими системами. Consul забезпечує такі функції:

- *Реєстр служб* - Consul дозволяє реєструвати служби в централізованому місці. Це дозволяє іншим службам знаходити одна одну.
- *Служба DNS* - Consul можна використовувати як службу DNS для розподілених систем. Це дозволяє службам знаходити одна одну за іменами.
- *Служба балансування навантаження* - Consul можна використовувати як службу балансування навантаження для розподілених систем. Це дозволяє рівномірно розподіляти навантаження між службами.
- *Служба відновлення після відмов* - Consul можна використовувати для відновлення після відмов розподілених систем. Це дозволяє відновити служби після збою.

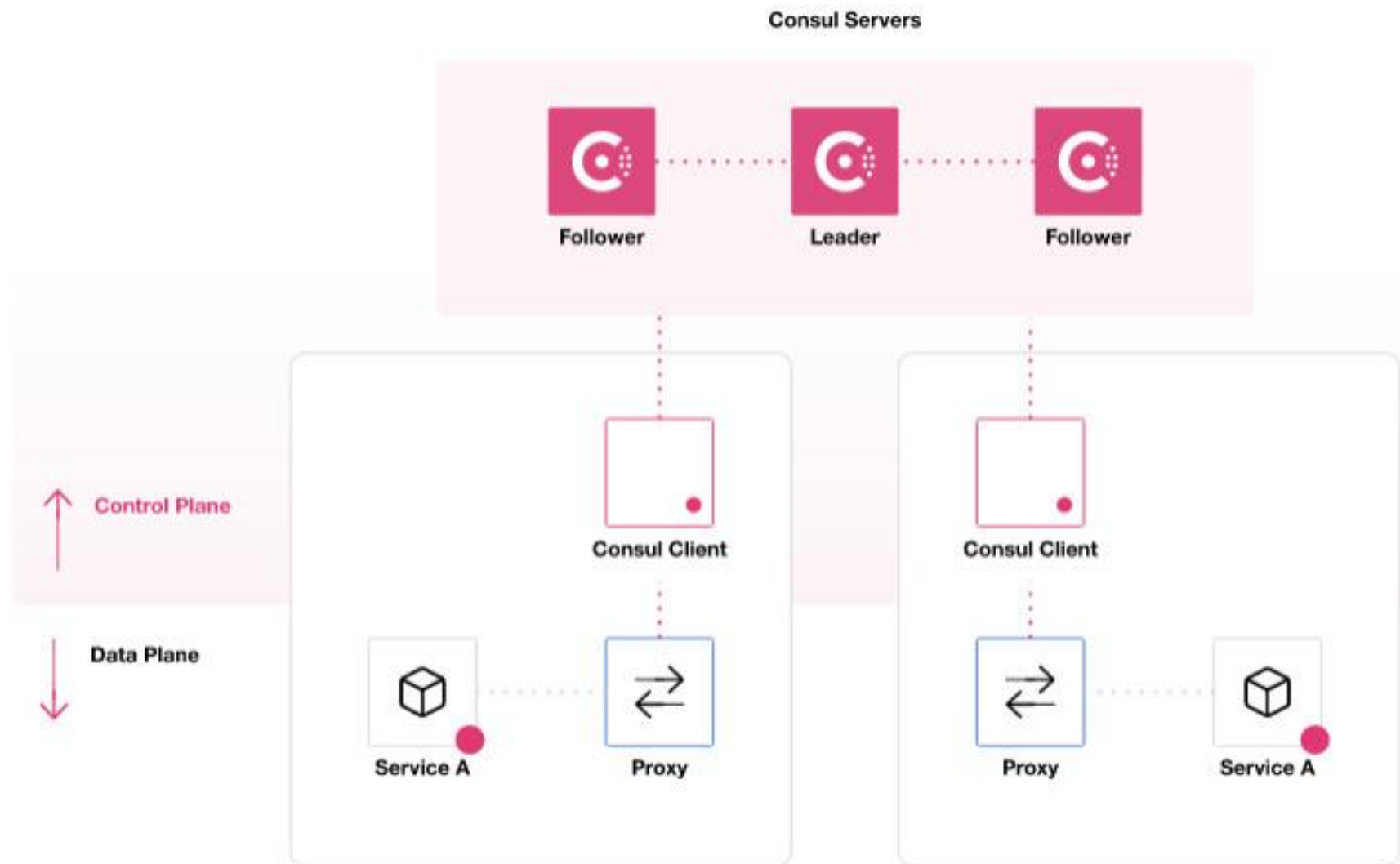




Архітектура Consul



Архітектура Consul





Datacenters

- ☐ Площина керування (control plane) Consul містить один або кілька центрів обробки даних.
- ☐ Центр обробки даних — це найменша одиниця інфраструктури Consul, яка може виконувати основні операції Consul.
- ☐ Центр обробки даних містить принаймні один агент сервера Consul, але розгортання в реальному світі містить три або п'ять агентів сервера та кілька агентів клієнта Consul.
- ☐ Можна створити кілька центрів обробки даних і дозволити вузлам у різних центрах даних взаємодіяти один з одним.



Кластер

- ☐ Набір агентів Consul, які знають один про одного, називається кластером.
- ☐ Терміни центр обробки даних і кластер часто використовуються як синоніми.
- ☐ Однак у деяких випадках кластер відноситься лише до агентів сервера Consul, наприклад у HCP Consul.
- ☐ В інших контекстах, як-от функція розділів адміністратора, включена в Consul Enterprise, кластер може означати колекцію клієнтських агентів.



Агент

- ☐ Можна запустити двійковий файл Consul з метою запуску агентів Consul, які є демонами, що реалізують функціональні можливості площини керування Consul.
- ☐ Можна запускати агентів як сервери або клієнти.



Серверний агент

- ❑ Агенти сервера Consul зберігають усю інформацію про стан, включаючи IP-адреси служб і вузлів, перевірки справності та конфігурацію.
- ❑ Рекомендується розгортати три або п'ять серверів у кластері. Чим більше серверів розгортається, тим більша стійкість і доступність у разі збою.
- ❑ Однак більша кількість серверів уповільнює роботу консенсусу, що є критично важливою функцією сервера, яка дозволяє Consul ефективно й результативно обробляти інформацію.



Протокол консенсусу₁

- ☐ Кластери консулів обирають єдиний сервер, який буде лідером, через процес, який називається консенсусом.
- ☐ Лідер обробляє всі запити та транзакції, що запобігає конфліктним оновленням у кластерах, що містять кілька серверів.
- ☐ Сервери, які в даний момент не виконують роль лідера кластера, називаються послідовниками (followers). Послідовники пересилають запити від клієнтських агентів лідеру кластеру.
- ☐ Лідер реплікує запити на всі інші сервери в кластері. Реплікація гарантує, що якщо лідер недоступний, інші сервери в кластері зможуть обрати іншого лідера без втрати даних.
- ☐ Сервери Consul встановлюють консенсус за допомогою алгоритму Raft на порту 8300.



Протокол консенсусу₂





Клієнтський агент

- ☐ Клієнти Consul повідомляють про стан справності вузла та служби в кластер Consul.
- ☐ У типовому розгортанні необхідно запускати клієнтські агенти на кожному обчислювальному вузлі у центрі обробки даних.
- ☐ Клієнти використовують віддалені виклики процедур (RPC) для взаємодії з серверами. За замовчуванням клієнти надсилають запити RPC на сервери на порту 8300.
- ☐ Немає обмежень щодо кількості клієнтських агентів або служб, які можна використовувати з Consul, але робочі розгортання повинні розподіляти послуги між кількома центрами обробки даних Consul.
- ☐ Використання розгортання з кількома центрами обробки даних підвищує стійкість інфраструктури та обмежує проблеми з площиною керування.
- ☐ Рекомендується розгортати максимум 5000 клієнтських агентів на центр обробки даних.

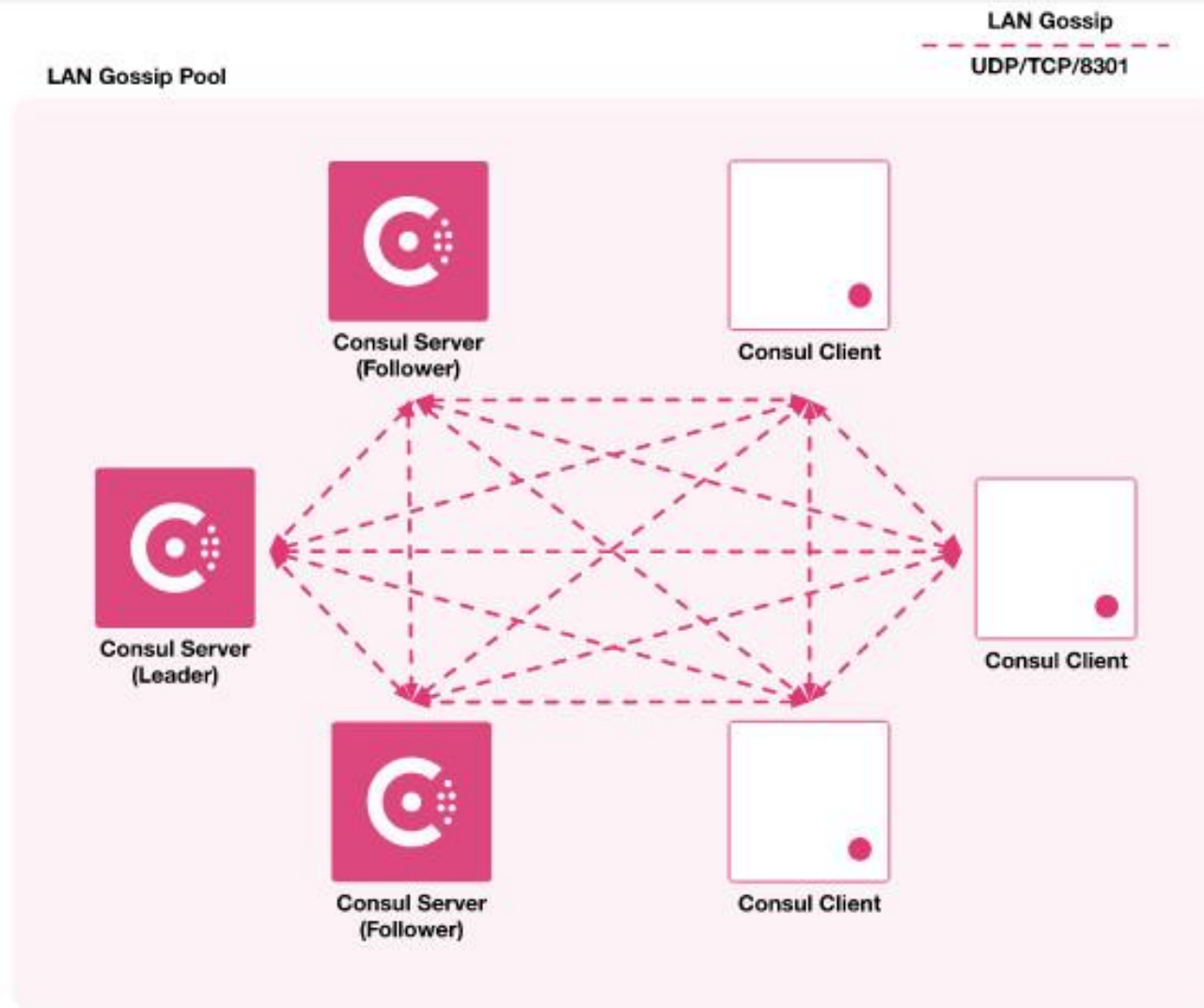


Пул пліток LAN (LAN Gossip pool)₁

- ☐ Агенти клієнта та сервера беруть участь у пулі пліток локальної мережі, щоб могли розповсюджувати та виконувати перевірку працездатності вузла.
- ☐ Агенти в пулі поширюють інформацію про перевірку працездатності в кластері.
- ☐ Зв'язок між агентами відбувається через порт 8301 за допомогою UDP. Agent gossip повертається до TCP, якщо UDP недоступний.



Пул пліток LAN (LAN Gossip pool)₂





Запити між датацентрами

- ☐ Кожен дата-центр Consul підтримує власний каталог послуг і їх працездатність.
- ☐ За замовчуванням інформація не копіюється між центрами обробки даних. WAN-федерація та кластерний піринг — це дві моделі розгортання кількох центрів обробки даних, які забезпечують підключення служб між центрами обробки даних.



WAN federation

- ❑ WAN-федерація – це підхід для з'єднання кількох центрів обробки даних Consul. Потрібно призначити основний центр обробки даних, який містить достовірну інформацію про всі центри обробки даних, включно з конфігураціями сітки служби та ресурсами списку контролю доступу (ACL).
- ❑ У цій моделі, коли агент клієнта запитує ресурс у віддаленому вторинному центрі обробки даних, локальний сервер Consul пересилає запит RPC на віддалений сервер Consul, який має доступ до ресурсу.
- ❑ Віддалений сервер надсилає результати на локальний сервер. Якщо віддалений центр обробки даних недоступний, його ресурси також недоступні. За замовчуванням об'єднані сервери WAN надсилають запити між центрами обробки даних через TCP на порт 8300.

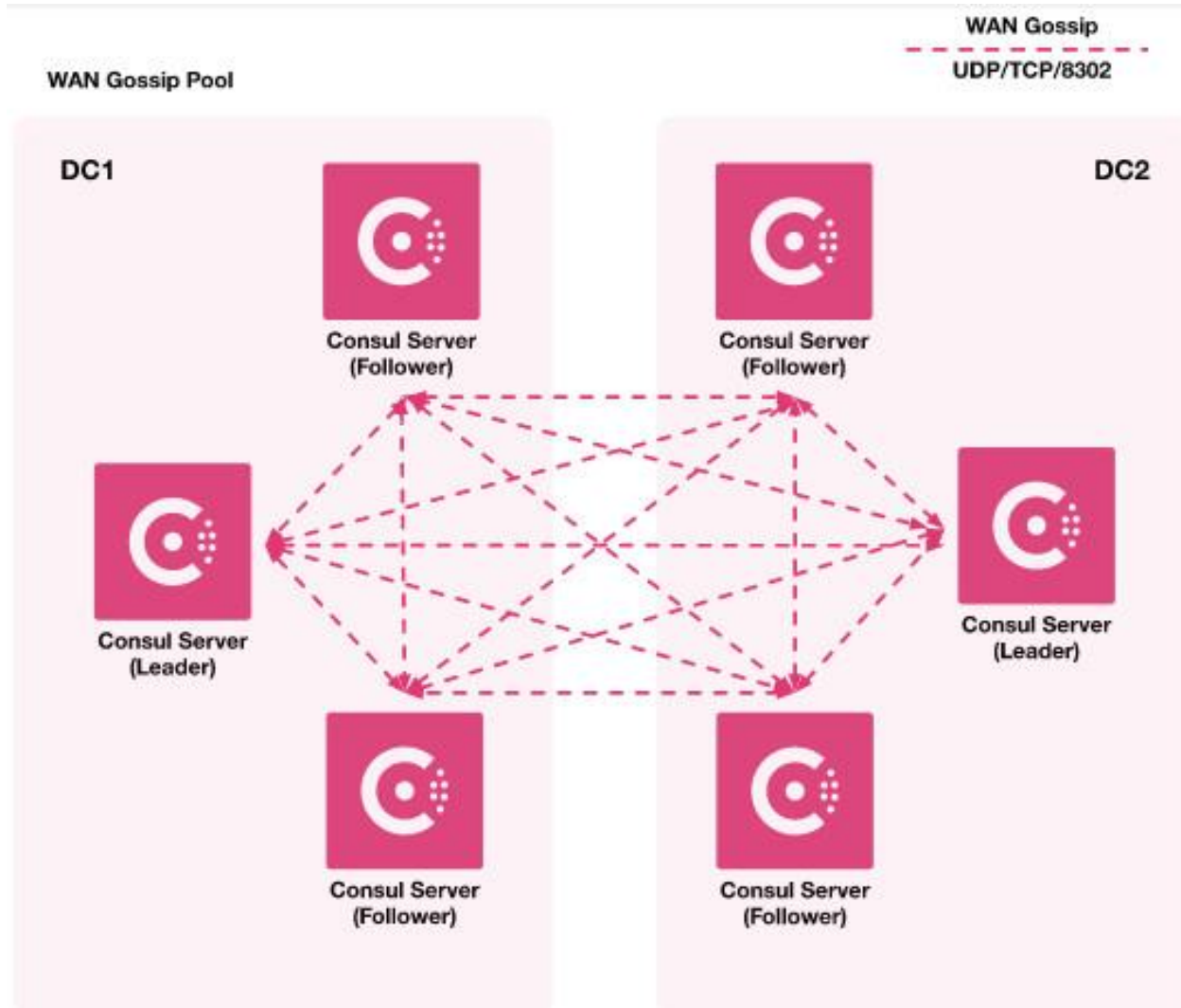


Пул пліток WAN (WAN Gossip pool)₁

- ☐ Сервери також можуть брати участь у пулі пліток глобальної мережі, оптимізованому для більшої затримки через Інтернет.
- ☐ Пул дозволяє серверам обмінюватися інформацією, такою як їхні адреси та справність, і витончено справлятися з втратою з'єднання у разі збою.
- ☐ На наведеній нижче схемі сервери в кожному центрі обробки даних беруть участь у пулі переговорів WAN, надсилаючи дані через TCP/UDP на порт 8302.



Пул пліток WAN (WAN Gossip pool)₂





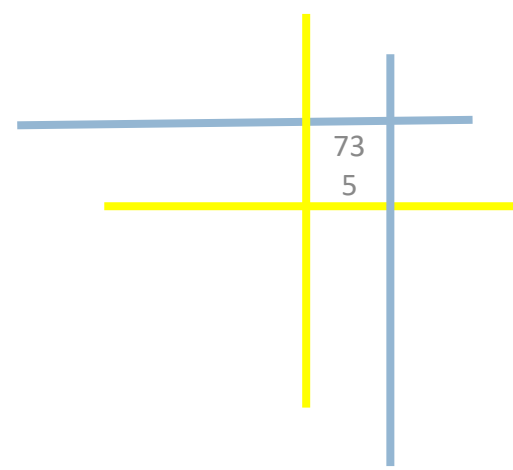
Кластерний піринг

- ☐ Можна створити однорангові з'єднання між двома або більше незалежними кластерами, щоб служби, розгорнуті в різних центрах обробки даних або адміністративних розділах, могли спілкуватися.
- ☐ Розділ адміністратора — це функція в Consul Enterprise, яка дає змогу визначати ізольовані мережеві регіони, які використовують однакові сервери Consul.
- ☐ У кластерній одноранговій моделі створюється маркер в одному з центрів обробки даних або розділів і налаштовується інший центр даних або розділ для представлення маркера для встановлення з'єднання.



Сховище ключ-значення

- ❑ Consul має вбудоване сховище типу ключ-значення, яке можна використовувати для зберігання конфігурацій та параметрів. Це дозволяє централізовано зберігати конфігурації та параметри, що робить їх доступними для всіх служб у розподіленій системі.
- ❑ Дані в сховищі Consul захищені за допомогою алгоритму шифрування AES-256. Консистентність даних забезпечується за допомогою алгоритму консенсусу RAFT.



Налаштування Consul



Інсталяція Consul₁

- ☐ Consul можна встановити на різних платформах, включаючи:
- Linux
 - Windows
 - macOS
 - Docker



Інсталяція Consul₂

❑ Для встановлення Consul на Linux потрібно виконати наступні кроки:

- Завантажте пакет Consul з офіційного веб-сайту.
- Розпакуйте пакет.
- Перейдіть до каталогу куди розпаковано пакет.
- Виконайте команду `consul agent -server` для запуску сервера Consul.



Інсталяція Consul₃

❑ Для встановлення Consul на Docker потрібно виконати наступні кроки:

- Створіть файл `Dockerfile` з таким змістом:

```
FROM consul:latest  
CMD ["consul", "agent", "-server"]
```

- Побудуйте образ Docker:

```
docker build -t consul .
```

- Запустіть контейнер Docker:

```
docker run -d -p 8500:8500 consul
```



Запуск серверів та агентів

- ❑ Для запуску серверів потрібно виконати команду `consul agent -server` на кожному сервері.
- ❑ Для запуску агентів потрібно виконати команду `consul agent` на кожному хості, де потрібно реєструвати службу.



Реєстрація та відкриття сервісів₁

❑ Для додавання нового сервісу в Consul потрібно виконати наступні кроки:

- Зареєструвати службу в агенті Consul.
- Оголосити службу в кластері Consul.

❑ Для реєстрації служби в агенті Consul потрібно виконати наступні дії:

- Налаштувати файл конфігурації агента Consul.
- Запустити агент Consul.



Реєстрація та відкриття сервісів₂

❑ У файлі конфігурації агента Consul потрібно вказати наступні параметри:

- `service_name`: назва служби.
- `service_tags`: теги служби.
- `service_address`: адреса служби.
- `service_port`: порт служби.

❑ Наприклад, наступний фрагмент файлу конфігурації агента Consul реєструє службу з назвою `"my-service"` на адресі `"192.168.1.10"` і порту 80:

```
service {  
  name = "my-service"  
  tags = ["web", "http"]  
  address = "192.168.1.10"  
  port = 80  
}
```



Реєстрація та відкриття сервісів₃

- ❑ Після реєстрації служби в агенті Consul потрібно оголосити службу в кластері Consul. Це можна зробити за допомогою команди `consul catalog register`.
- ❑ Наприклад, наступна команда оголошує службу з назвою "my-service" в кластері Consul:

```
consul catalog register my-service
```
- ❑ Після реєстрації та оголошення служби її можна отримати доступ за допомогою DNS або HTTP.

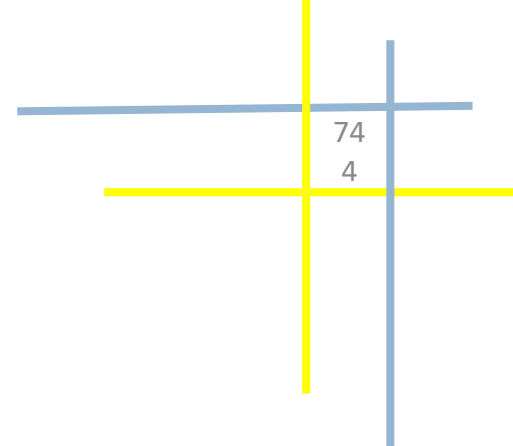


Реєстрація та відкриття сервісів₄

- ❑ Consul підтримує службу DNS, яка дозволяє отримувати доступ до служб за іменами. Для цього потрібно вказати ім'я служби в DNS-запиті.
- ❑ Наприклад, наступний DNS-запит поверне адресу служби "my-service":

```
dig my-service
```
- ❑ Consul також підтримує службу HTTP, яка дозволяє отримувати доступ до служб за HTTP-адресами. Для цього потрібно вказати HTTP-адресу служби в HTTP-запиті.
- ❑ Наприклад, наступний HTTP-запит поверне відповідь від служби "my-service":

```
curl http://192.168.1.10:80
```



Захист Consul



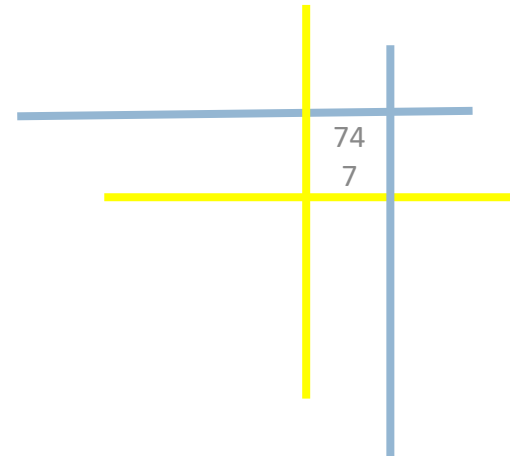
Захист Consul₁

- ☐ Для захисту Consul від несанкціонованого доступу потрібно встановити базове забезпечення безпеки, яке включає в себе:
- ☐ *Налаштування паролів і сертифікатів*: для всіх компонентів Consul потрібно налаштувати паролі та сертифікати. Це допоможе захистити доступ до Consul від несанкціонованих користувачів.
- ☐ *Використання firewall*: для захисту Consul від несанкціонованого доступу ззовні потрібно використовувати firewall. Це допоможе блокувати несанкціоновані з'єднання з Consul.



Захист Consul₂

- ❑ Consul підтримує шифрування TLS для комунікацій між компонентами. Для шифрування комунікацій між компонентами потрібно створити сертифікати TLS і налаштувати їх використання в Consul.
- ❑ Consul також підтримує інші методи безпеки, такі як аутентифікація на основі ролей (RBAC) і контроль доступу на основі ролей (RBAC).
- ❑ Consul можна використовувати з іншими системами безпеки, такими як Kubernetes і HashiCorp Vault.



Моніторинг Consul



Відслідковування здоров'я сервісів₁

- ❑ Consul можна використовувати для відслідковування стану сервісів. Для цього потрібно вказати агенту Consul, як перевіряти стан служби.
- ❑ Consul підтримує різні види перевірок здоров'я, такі як:
 - HTTP-перевірка: перевіряє, чи можна отримати доступ до служби за допомогою HTTP-запиту.
 - TCP-перевірка: перевіряє, чи можна з'єднатися з службою за допомогою TCP-з'єднання.
 - Простий перевірочний код: перевіряє, чи повертає служба певний код стану.



Відслідковування здоров'я сервісів₂

□ Для налаштування перевірки здоров'я потрібно вказати наступні параметри:

- `check_name`: назва перевірки.
- `check_interval`: інтервал перевірки.
- `check_timeout`: час очікування для перевірки.
- `check_method`: метод перевірки.
- `check_args`: аргументи для перевірки.

□ Наприклад, наступний фрагмент файлу конфігурації агента Consul налаштовує HTTP-перевірку для служби "my-service":

```
check {  
  name = "my-service"  
  interval = "10s"  
  timeout = "2s"  
  method = "http"  
  args = ["/status"]  
}
```



Відслідковування здоров'я сервісів₃

- ☐ Consul автоматично виявляє недоступні сервіси, якщо вони не проходять перевірки здоров'я.
- ☐ Якщо служба не проходить перевірку здоров'я, її статус змінюється на "недоступний".



Налаштування правил тривоги₁

❑ Consul дозволяє налаштовувати правила тривоги, які запускаються при зміні стану сервісів. Правила тривоги можна використовувати для сповіщення про такі події, як:

- Недоступність сервісу
- Зміна критичності стану сервісу
- Зміна кількості доступних серверів для сервісу



Налаштування правил тривоги₂

- ❑ Consul дозволяє сповіщати про зміни стану сервісів за допомогою різних каналів, таких як:
 - *Email*: Consul може надсилати сповіщення по електронній пошті на вказані адреси.
 - *SMS*: Consul може надсилати сповіщення SMS на вказані номери телефонів.
 - *Webhook*: Consul може надсилати сповіщення на вказані веб-адреси.



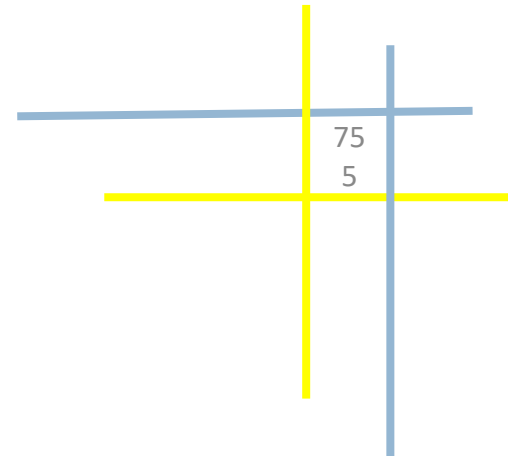
Використання метрик та логів для моніторингу стану Consul

- ☐ Consul збирає метрики та логи для моніторингу стану системи. Метрики - це числові значення, які вимірюють різні аспекти системи. Логи - це текстові записи, які містять інформацію про події, що відбуваються в системі.
- ☐ Consul збирає такі метрики:
 - *Статус сервісів*: статус, кількість доступних серверів, час відповіді
 - *Використання ресурсів*: використання CPU, пам'яті, диску
 - *Виконання перевірок здоров'я*: час виконання, кількість помилок
- ☐ Consul збирає такі логи:
 - *Журнал перевірок здоров'я*: повідомлення про результати перевірок здоров'я
 - *Журнал подій*: повідомлення про різні події, що відбуваються в системі



Аналіз даних для вдосконалення процесів

- ❑ Дані, зібрані Consul, можна використовувати для аналізу стану системи. Цей аналіз можна використовувати для вдосконалення процесів управління розподіленими системами.
- ❑ Наприклад, аналіз даних можна використовувати для:
 - *Виявлення проблем*: виявлення проблем у системі, таких як недоступні сервіси або високе використання ресурсів
 - *Покращення ефективності*: покращення ефективності системи, наприклад, шляхом оптимізації використання ресурсів
 - *Прогнозування майбутніх подій*: прогнозування майбутніх подій, таких як можливі збої в системі



Інтеграція з іншими інструментами



Інтеграція з іншими інструментами

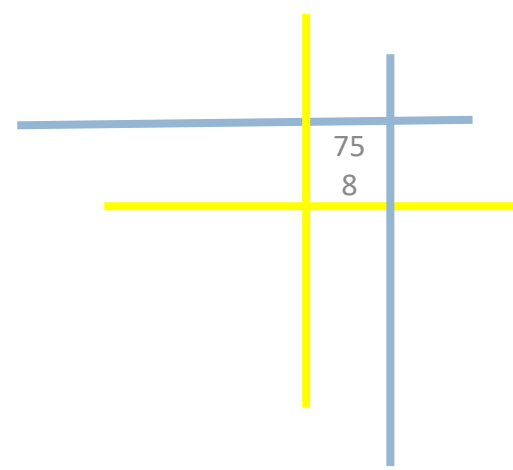
❑ Consul можна інтегрувати з іншими інструментами моніторингу та управління, такими як:

- *Prometheus*: Consul можна використовувати як джерело даних для Prometheus. Це дозволяє використовувати Prometheus для моніторингу стану сервісів, що зареєстровані в Consul.
- *Grafana*: Consul можна використовувати як джерело даних для Grafana. Це дозволяє створювати графіки та інші візуальні елементи для відображення даних, зібраних Consul.
- *ELK*: Consul можна використовувати як джерело даних для ELK. Це дозволяє використовувати ELK для зберігання та аналізу логів, зібраних Consul.



Взаємодія з Kubernetes та іншими оркестраторами

- ❑ Consul можна використовувати з Kubernetes та іншими оркестраторами для автоматизації розгортання та управління розподіленими системами.
- ❑ Наприклад, Consul можна використовувати для:
 - *Автоматичного розгортання сервісів*: Consul можна використовувати для автоматичного розгортання сервісів у Kubernetes. Це дозволяє швидко та легко розгортати нові сервіси у розподіленій системі.
 - *Автоматичного виявлення та масштабування сервісів*: Consul можна використовувати для автоматичного виявлення та масштабування сервісів у Kubernetes. Це дозволяє забезпечити доступність і масштабованість розподіленої системи.



Список рекомендованих джерел для самопідготовки



Список рекомендованих джерел для самопідготовки₁

1. Rist, Oliver. "Getting Started with VirtualBox." Packt Publishing, 2018. (190 pages)
2. Preece, Robert. "VirtualBox 6.0: Beginners Guide on how to Install, Configure, and Use VirtualBox." Independently published, 2019. (70 pages)
3. Ragar, Victor. "Mastering VirtualBox: Expert techniques for Linux and Windows virtualization, 2nd Edition." Packt Publishing, 2016. (342 pages)
4. Russell, Dusty. "Oracle VM VirtualBox: Beginner's Guide." Packt Publishing, 2018. (284 pages)
5. Russell, Dusty. "Oracle VM VirtualBox 4.0 on OS X Snow Leopard." CreateSpace Independent Publishing Platform, 2010. (128 pages)
6. Herter, Nicholas. "VirtualBox 6.0 – The Ultimate Beginner's Guide." Independently published, 2019. (97 pages)
7. Silverman, Steven. "Oracle VM VirtualBox: An Ultimate Guide Book on Oracle Virtualization Essentials." Independently published, 2018. (87 pages)



Список рекомендованих джерел для самопідготовки₂

8. Version Control Systems - GeeksforGeeks. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/version-control-systems/> (дата звернення: 21.09.2023).
9. What Is Git | Explore A Distributed Version Control Tool | Edureka. Edureka. URL: <https://www.edureka.co/blog/what-is-git/> (дата звернення: 21.09.2023).
10. How to Git(Hub) Sponge 8.0.0. Sponge Documentation. URL: <https://docs.spongepowered.org/stable/ru/contributing/howtogit.html> (дата звернення: 21.09.2023).
11. Підручник з робочого столу GitHub - Співпрацюйте з GitHub зі свого робочого столу - Інший. Огляди, Ігри, Розваги, Листопад 2022. URL: <https://uk.myservername.com/github-desktop-tutorial-collaborate-with-github-from-your-desktop> (дата звернення: 21.09.2023).
12. Що таке Git та Github – керівництво для початківців. SebWeo. URL: <https://sebweo.com/scho-take-git-ta-github-kerivnitstvo-dlya-pochatktivtsiv/> (дата звернення: 21.09.2023).
13. The Git experience in Visual Studio. Microsoft Learn: Build skills that open doors in your career. URL: <https://learn.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2022> (дата звернення: 21.09.2023).



Список рекомендованих джерел для самопідготовки₃

14. Hashimoto, Hashicorp. "Vagrant: Up and Running." O'Reilly Media, 2013. (160 pages)
15. Mitchell, Hashimoto. "Vagrant: The Definitive Guide." O'Reilly Media, 2015. (378 pages)
16. Burkov, Andrey. "Vagrant Cookbook." Packt Publishing, 2015. (284 pages)
17. Henningsson, Magnus. "Vagrant Virtual Development Environment Cookbook." Packt Publishing, 2015. (288 pages)
18. Carless, Thomas. "Vagrant Virtual Development Environment Cookbook - Second Edition." Packt Publishing, 2017. (322 pages)
19. Verminski, Cody. "Vagrant Virtual Development Environment Cookbook - Third Edition." Packt Publishing, 2018. (364 pages)
20. Hashimoto, Mitchell, McCune. "Vagrant: Up and Running - Second Edition." O'Reilly Media, 2017. (382 pages)



Список рекомендованих джерел для самопідготовки₄

21. capistrano/capistrano: A deployment automation tool built on Ruby, Rake, and SSH [Електронний ресурс]. – Режим доступу: <https://github.com/capistrano/capistrano> (дата звернення: 8.11.2023 р.).
22. mhutter/capistrano-recipes: A collection of recipes around Rails [Електронний ресурс]. – Режим доступу: <https://github.com/mhutter/capistrano-recipes> (дата звернення: 8.11.2023 р.).
23. Burgess, R. (2012). "Capistrano Recipes: Structure, Design, and Modular Automation for Web Developers." Pragmatic Bookshelf.
24. Matsumoto, Y. (2008). "The Ruby Programming Language." O'Reilly Media.



Список рекомендованих джерел для самопідготовки₅

25. Chef Software DevOps Automation Solutions | Chef [Електронний ресурс]. – Режим доступу: <https://www.chef.io/> (дата звернення: 6.12.2023 р.).

26. Chef Community Cookbooks [Електронний ресурс]. – Режим доступу: <https://github.com/chef-cookbooks/> (дата звернення: 6.12.2023 р.).

27. Chef Supermarket [Електронний ресурс]. – Режим доступу: <https://supermarket.chef.io/> (дата звернення: 6.12.2023 р.).

28. Chef Infrastructure Automation Cookbook [Електронний ресурс]. – Режим доступу: <https://github.com/manikabedi/devops-batch/blob/master/Chef%20Infrastructure%20Automation%20Cookbook,%20Second%20Edition.pdf> (дата звернення: 6.12.2023 р.).



Список рекомендованих джерел для самопідготовки₆

29. Puppet Infrastructure & IT Automation at Scale | Puppet by Perforce [Електронний ресурс]. – Режим доступу: <https://www.puppet.com/> (дата звернення: 9.12.2023 р.).

30. Vagrant and Docker followed by Puppet to provision complex environments [Електронний ресурс]. – Режим доступу: <https://technology.amis.nl/continuous-delivery/vagrant-and-docker-followed-by-puppet-to-provision-complex-environments/> (дата звернення: 9.12.2023 р.).

31. How to Install Puppet on Ubuntu 20.04 [Електронний ресурс]. – Режим доступу: <https://phoenixnap.com/kb/install-puppet-ubuntu> (дата звернення: 9.12.2023 р.).

32. Installing Puppet Modules – Librarian Puppet [Електронний ресурс]. – Режим доступу: <https://aggarwalarpit.wordpress.com/2015/11/07/installing-puppet-modules-librarian-puppet/> (дата звернення: 9.12.2023 р.).

33. Puppetlabs/docker. Module for installing and managing docker. Puppet Forge [Електронний ресурс]. – Режим доступу: <https://forge.puppet.com/modules/puppetlabs/docker/readme> (дата звернення: 9.12.2023 р.).



Список рекомендованих джерел для самопідготовки

34. Ansible is simple IT Automation [Електронний ресурс]. – Режим доступу: <https://www.ansible.com/> (дата звернення: 20.12.2023 р.).

35. Chapter 8 - Ansible Cookbooks [Електронний ресурс]. – Режим доступу: <https://bjrcjr.github.io/pdfs/devops/ansible-cookbooks.pdf> (дата звернення: 20.12.2023 р.).

36. Ansible playbooks [Електронний ресурс]. – Режим доступу: https://docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html (дата звернення: 20.12.2023 р.).

37. WordPress+phpMyAdmin+Nginx+PHP-FPM(Apache)+proftpd+ansible+docker [Електронний ресурс]. – Режим доступу: <https://github.com/amihura/ansible-docker-lamp> (дата звернення: 20.12.2023 р.).

38. Vagrant Centos 7 lamp using Ansible playbook [Електронний ресурс]. – Режим доступу: <https://github.com/skecskes/vagrant-centos7-ansible-lamp/tree/master> (дата звернення: 20.12.2023 р.).



Список рекомендованих джерел для самопідготовки₈

39. Kubernetes. Part I: Overview [Електронний ресурс]. – Режим доступу: <https://blog.ipeacocks.info/2017/08/kubernetes-part-i-overview.html> (дата звернення: 24.12.2023 р.).

40. A Guide to Kubernetes Etcd: All You Need to Know to Set up Etcd Clusters [Електронний ресурс]. – Режим доступу: <https://superuser.openinfra.dev/articles/a-guide-to-kubernetes-etcd-all-you-need-to-know-to-set-up-etcd-clusters/> (дата звернення: 24.12.2023 р.).

42. Prometheus. From Metrics To Insight [Електронний ресурс]. – Режим доступу: <https://blog.ipeacocks.info/2018/10/prometheus-from-metrics-to-insight.html> (дата звернення: 24.12.2023 р.).

42. Consul Architecture [Електронний ресурс]. – Режим доступу: <https://developer.hashicorp.com/consul/docs/architecture> (дата звернення: 24.12.2023 р.).