



Research & Development Team

Microservices with Spring Boot

www.pnpsw.com

sommai.k@gmail.com

081-754-4663

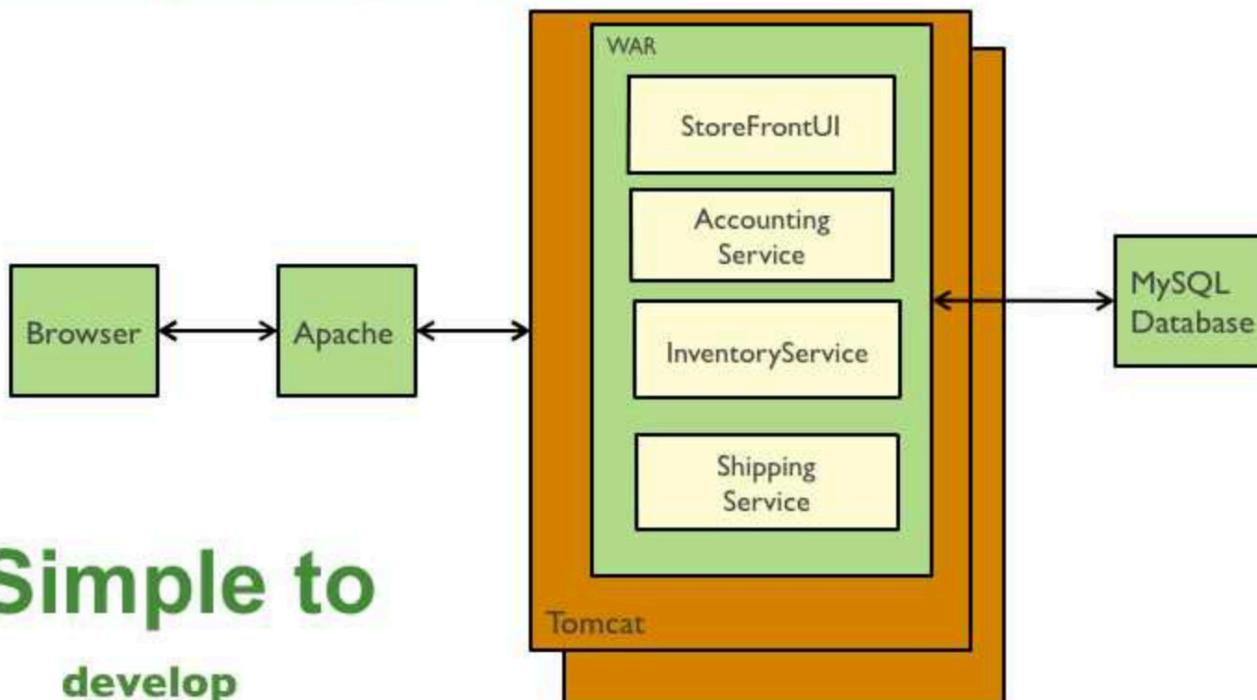
Lineid : sommaik

Introduction to

MICROSERVICE ARCHITECTURE

Monolithic Architecture

Traditional web application architecture



Simple to

develop
test
deploy
scale

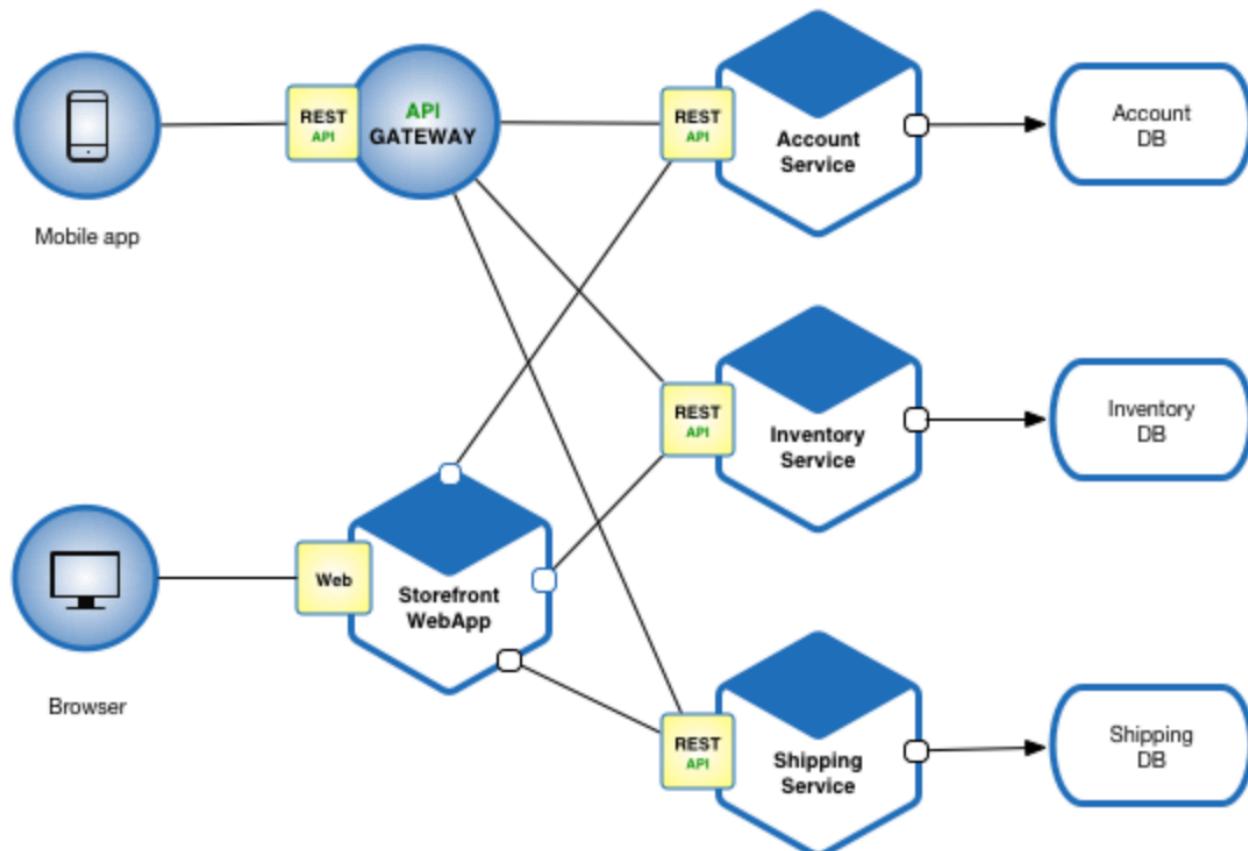
Benefits

- Simple to develop - the goal of current development tools and IDEs is to support the development of monolithic applications
- Simple to deploy - you simply need to deploy the WAR file (or directory hierarchy) on the appropriate runtime
- Simple to scale - you can scale the application by running multiple copies of the application behind a load balancer

Drawbacks

- The large monolithic code base intimidates developers, especially ones who are new to the team.
- Overloaded IDE
- Overloaded web container
- Continuous deployment is difficult
- Scaling the application can be difficult
- Obstacle to scaling development
- Requires a long-term commitment to a technology stack

Microservice Architecture



Benefits

- Enables the continuous delivery and deployment of large, complex applications.
 - Improved maintainability - each service is relatively small and so is easier to understand and change
 - Better testability - services are smaller and faster to test
 - Better deployability - services can be deployed independently
 - It enables you to organize the development effort around multiple, autonomous teams. Each (so called two pizza) team owns and is responsible for one or more services. Each team can develop, test, deploy and scale their services independently of all of the other teams.

Benefits #2

- Each microservice is relatively small:
 - Easier for a developer to understand
 - The IDE is faster making developers more productive
 - The application starts faster, which makes developers more productive, and speeds up deployments
- Improved fault isolation. For example, if there is a memory leak in one service then only that service will be affected. The other services will continue to handle requests. In comparison, one misbehaving component of a monolithic architecture can bring down the entire system.
- Eliminates any long-term commitment to a technology stack. When developing a new service you can pick a new technology stack. Similarly, when making major changes to an existing service you can rewrite it using a new technology stack.

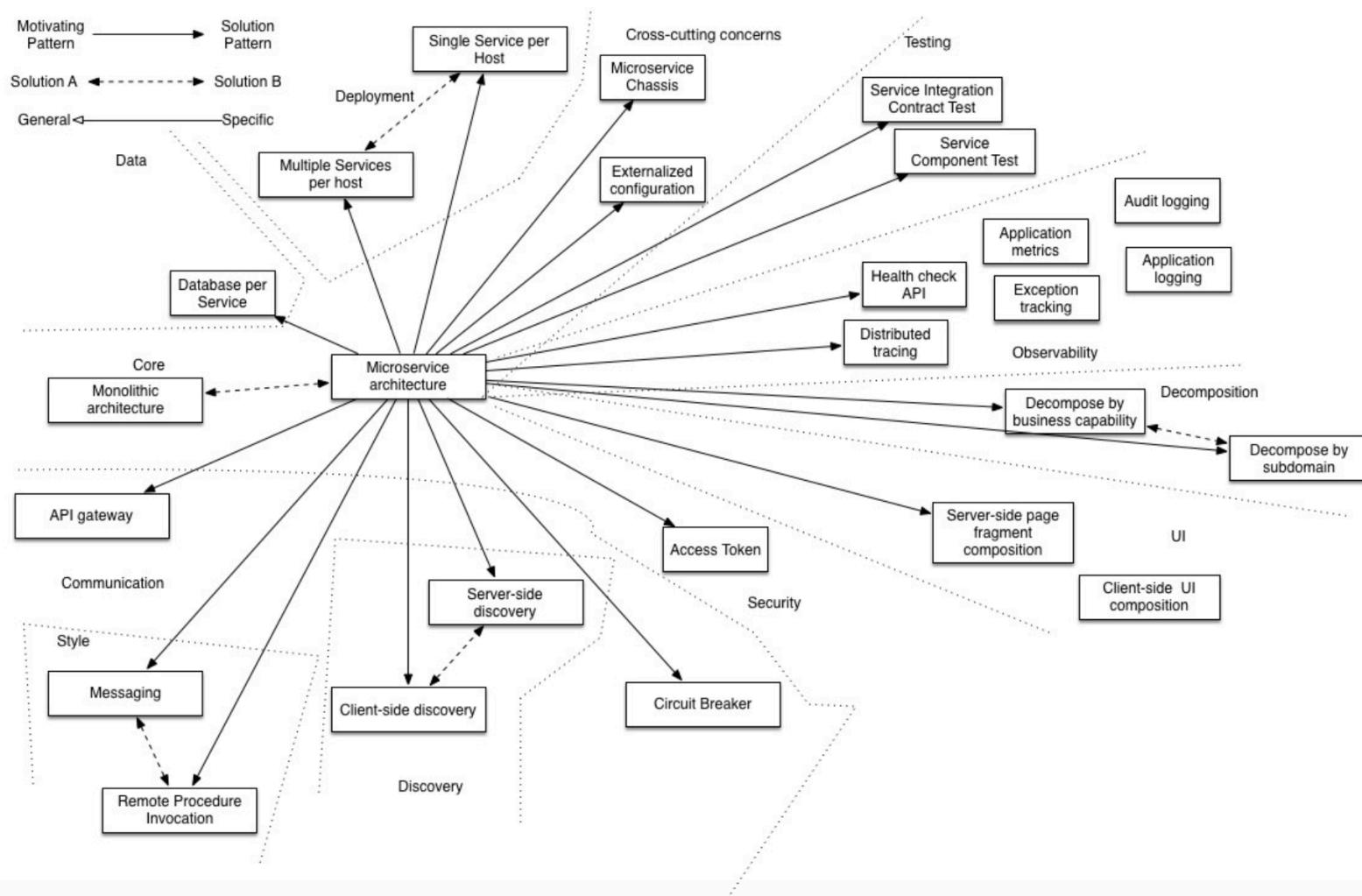
How to decompose

- Decompose by business capability and define services corresponding to business capabilities.
- Decompose by domain-driven design subdomain.
- Decompose by verb or use case and define services that are responsible for particular actions. e.g. a Shipping Service that's responsible for shipping complete orders.
- Decompose by nouns or resources by defining a service that is responsible for all operations on entities/resources of a given type. e.g. an Account Service that is responsible for managing user accounts.

Drawbacks

- Developers must deal with the additional complexity of creating a distributed system:
 - Developers must implement the inter-service communication mechanism and deal with partial failure
 - Implementing requests that span multiple services is more difficult
 - Testing the interactions between services is more difficult
 - Implementing requests that span multiple services requires careful coordination between the teams
 - Developer tools/IDEs are oriented on building monolithic applications and don't provide explicit support for developing distributed applications.
- Deployment complexity. In production, there is also the operational complexity of deploying and managing a system comprised of many different services.

Related patterns



Software

INSTALLATION

การติดตั้ง

JDK 8

การติดตั้ง JDK 8

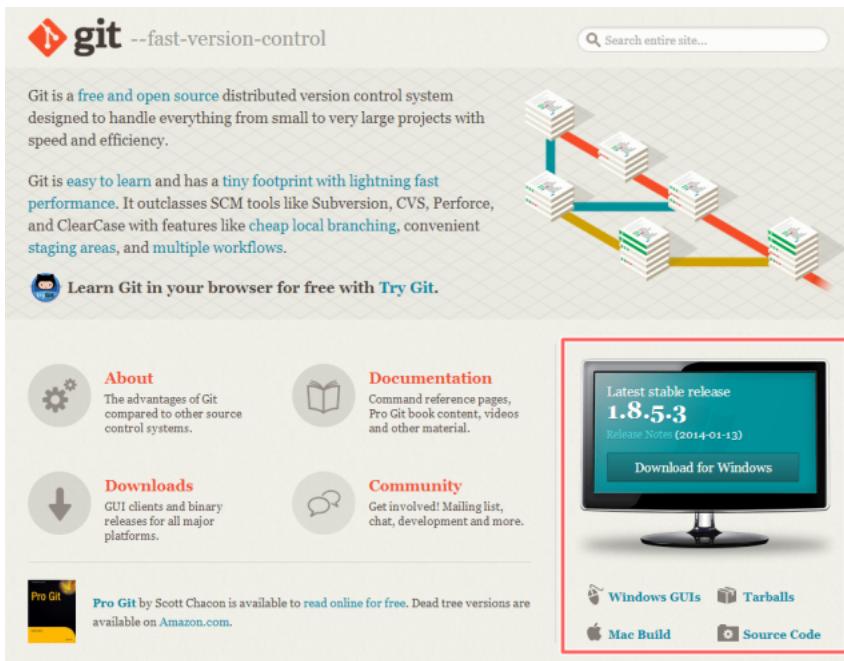
- download JDK 8 จาก link ดังนี้
 - <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
- ติดตั้งตามขั้นตอนไปจนสิ้นสุดการติดตั้ง
 - ทดสอบการติดตั้งโดยการ Check java version
 - java –version

การติดตั้ง

GIT FOR WINDOWS

การติดตั้ง GIT สำหรับ Windows

- เข้า website <https://git-scm.com/downloads>
- เลือก Download Git เพื่อติดตั้งบน Windows



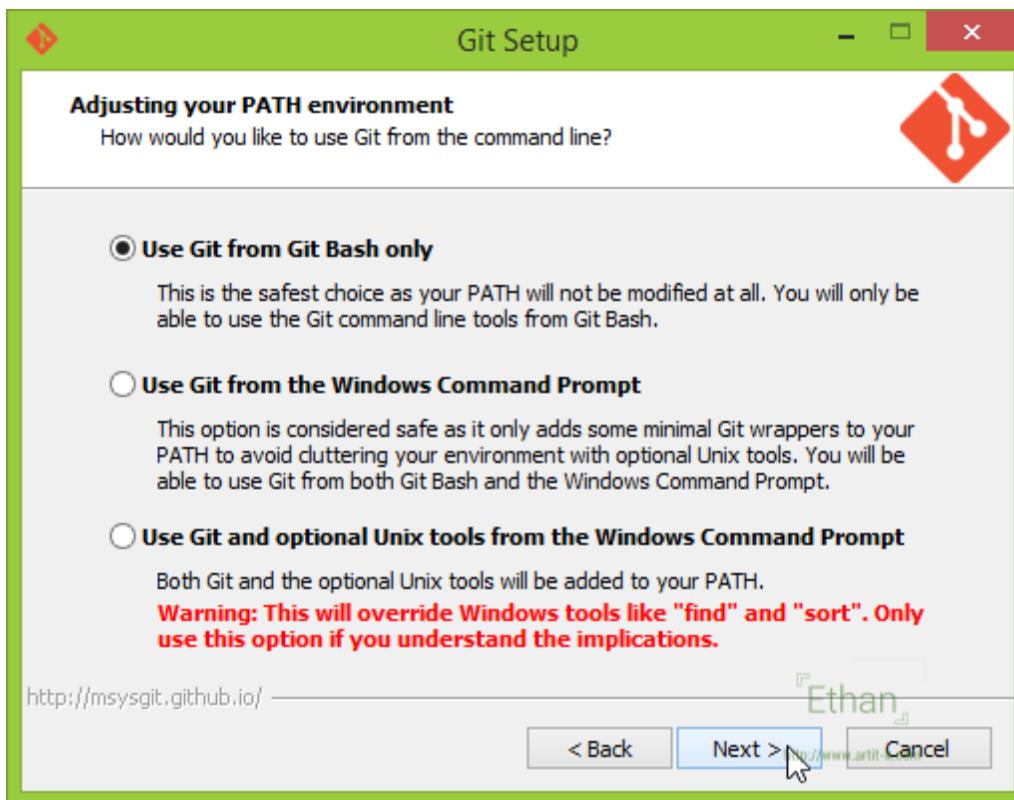
การติดตั้ง GIT สำหรับ Windows #2

- ติดตั้งโดยใช้สิทธิ Administrator ในการติดตั้ง



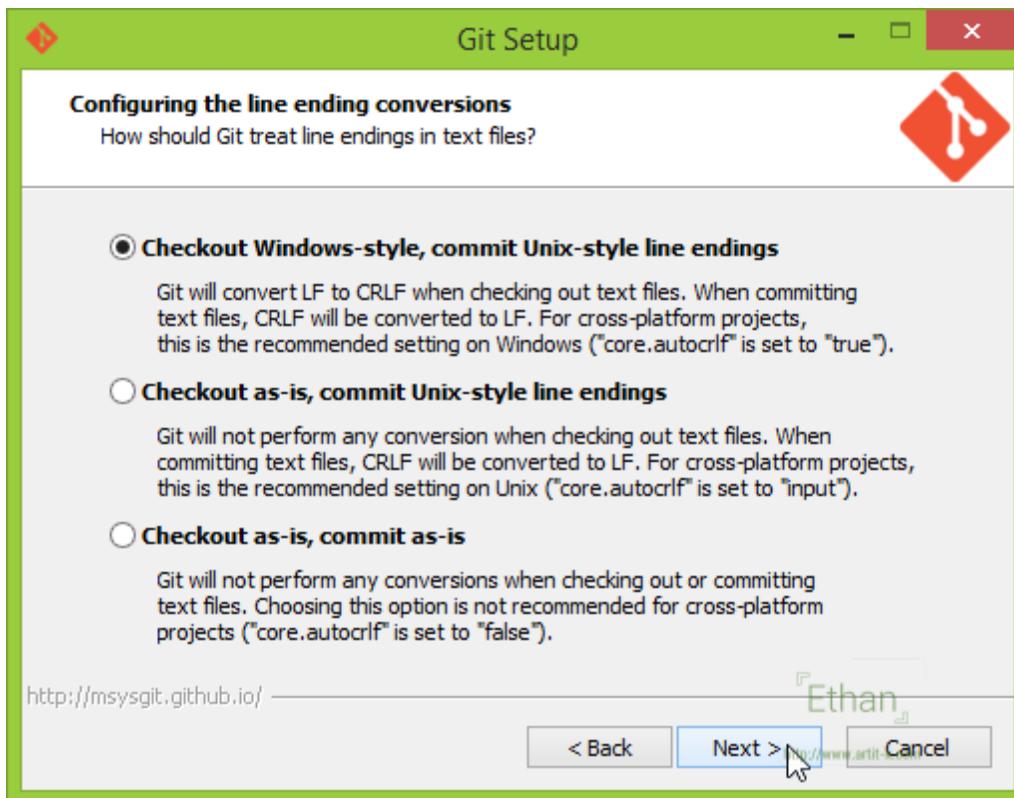
การติดตั้ง GIT สำหรับ Windows #3

- เลือกติดตั้งแบบ **Use Git from the Windows Command Prompt**



การติดตั้ง GIT สำหรับ Windows #4

- เลือกเป็น **Checkout Windows-style, commit Unix-style line endings**



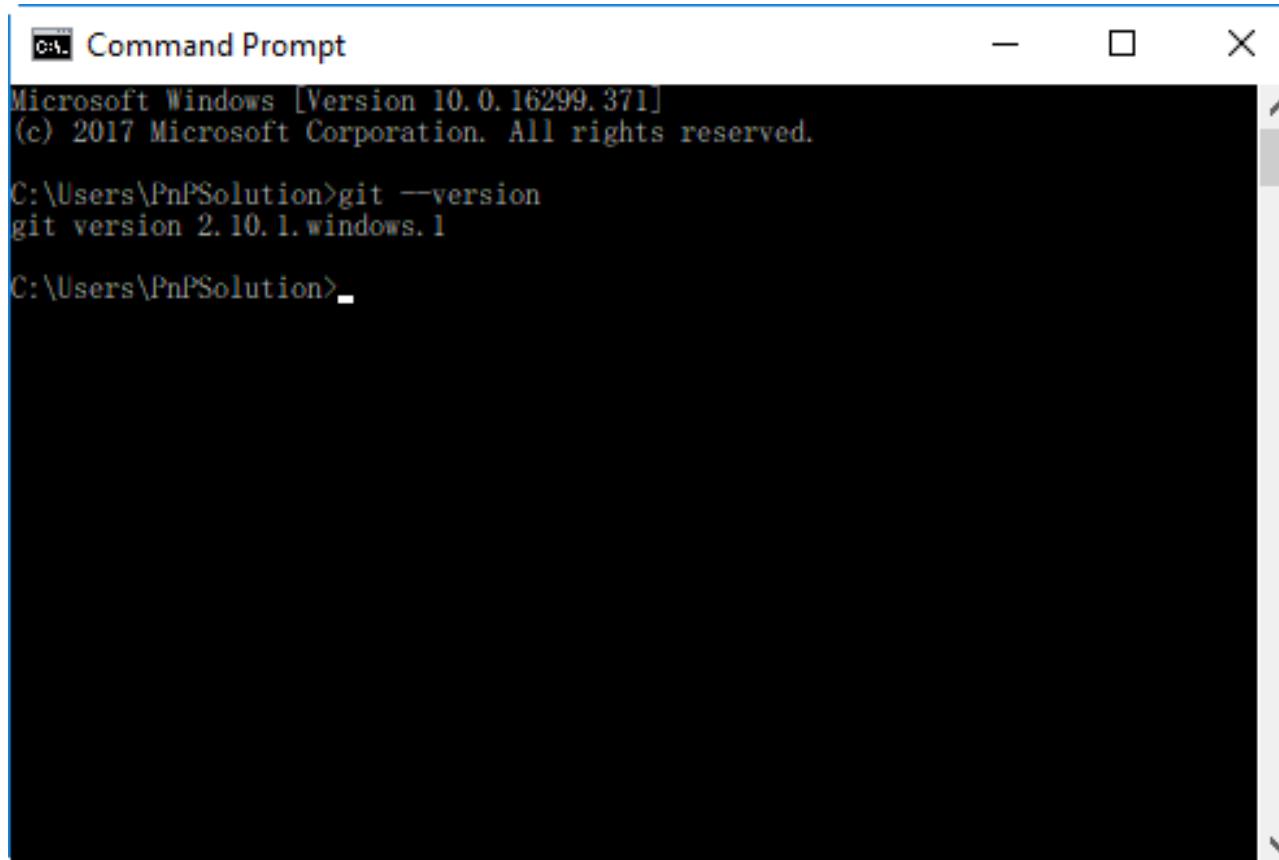
การติดตั้ง GIT สำหรับ Windows #5

- กดปุ่ม next ไปจนถึงหน้าสุดท้าย



ทดสอบหลังการติดตั้ง Git

- เปิดโปรแกรม cmd และพิมพ์คำสั่ง git --version



```
Microsoft Windows [Version 10.0.16299.371]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\PnP\Documents>git --version
git version 2.10.1.windows.1

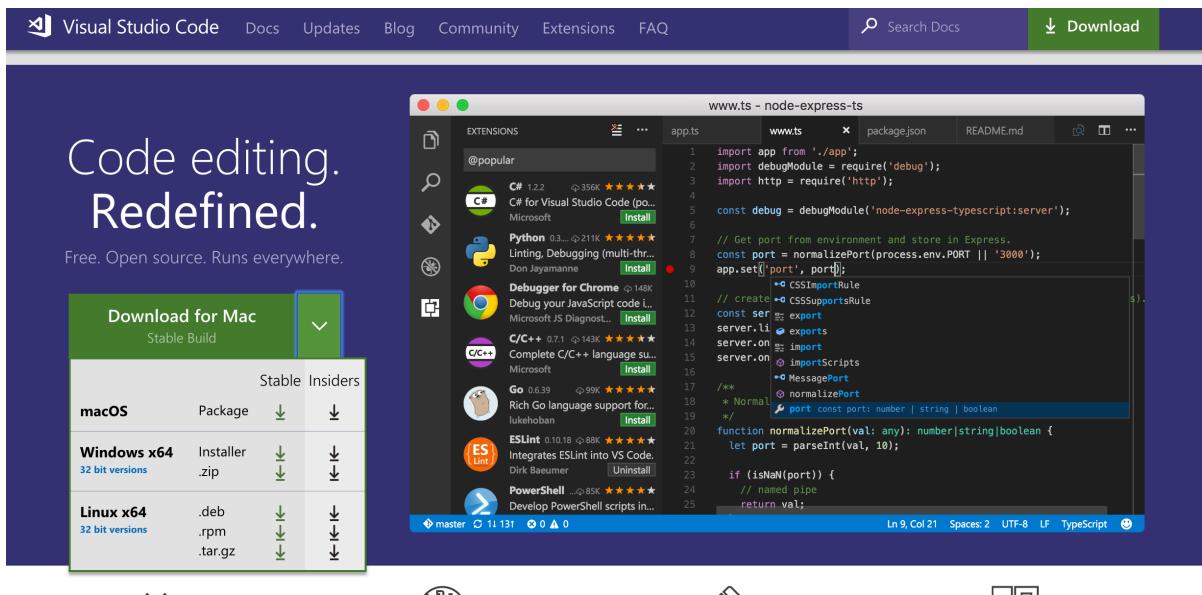
C:\Users\PnP\Documents>
```

การติดตั้ง

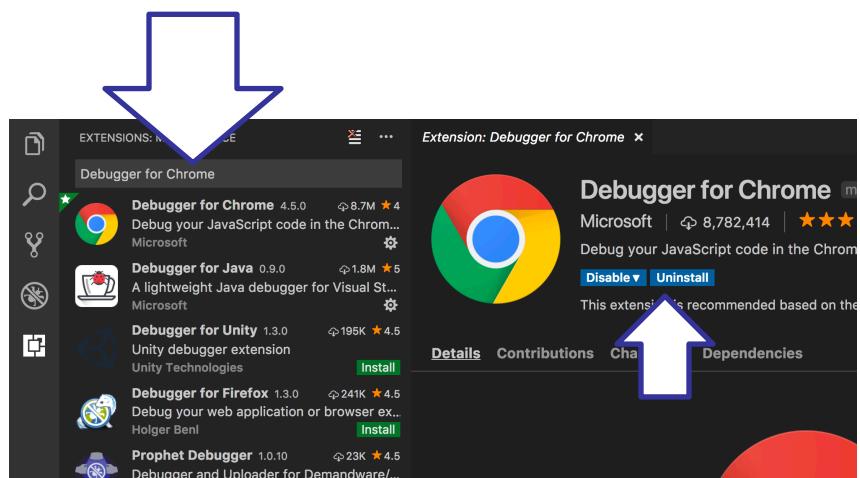
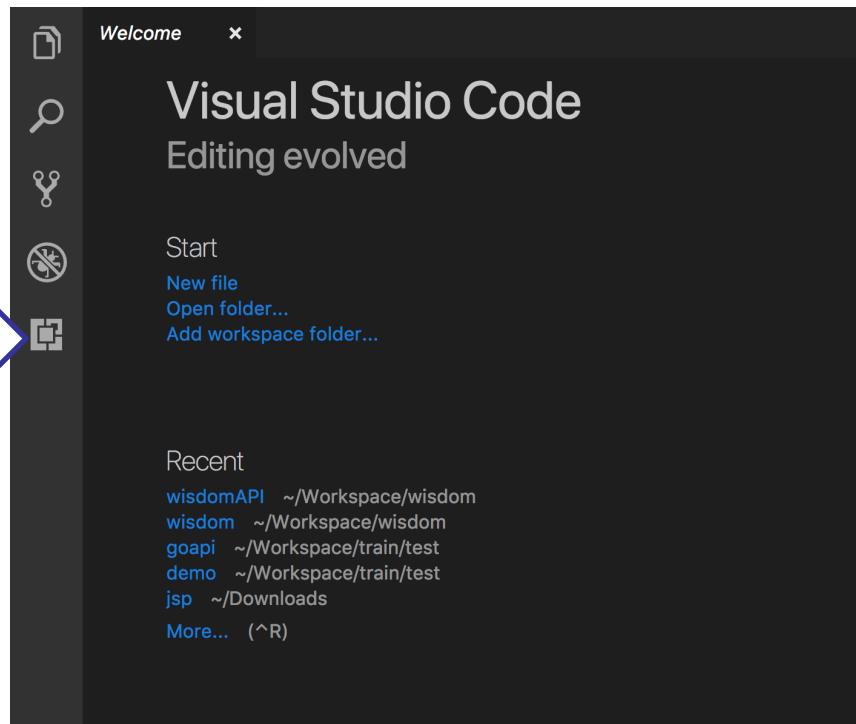
VISUAL STUDIO CODE

การติดตั้ง VSC

- เข้าไปที่ website <https://code.visualstudio.com/>
- เลือก download สำหรับ windows (stable)



Install Extensions



Install Visual Studio Code Extensions

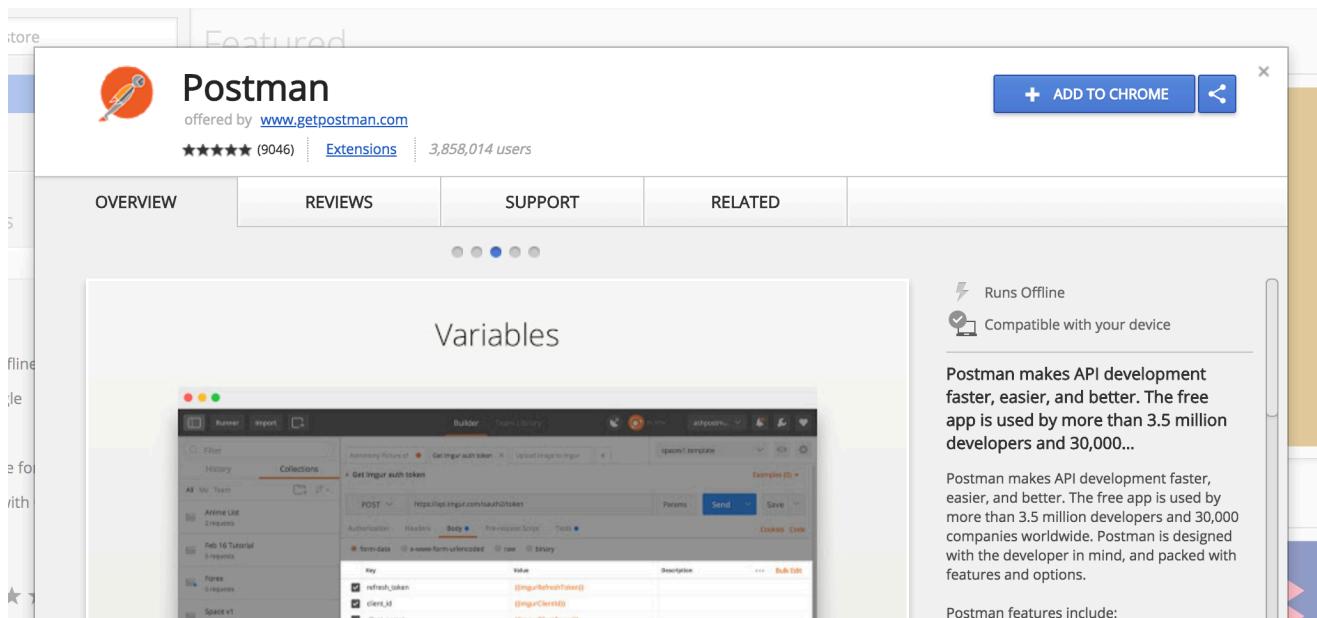
- Java Extension Pack [[vscjava.vscode-java-pack](#)]
- Spring Boot Tools [[Pivotal.vscode-spring-boot](#)]
- Spring Initializr Java Support [[vscjava.vscode-spring-initializr](#)]
- Bracket Pair Colorizer [[coenraads.bracket-pair-colorizer](#)]

การติดตั้ง

POSTMAN

การติดตั้ง postman

- เข้า url <https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjb dggehcdcbncddd domop?hl=en>

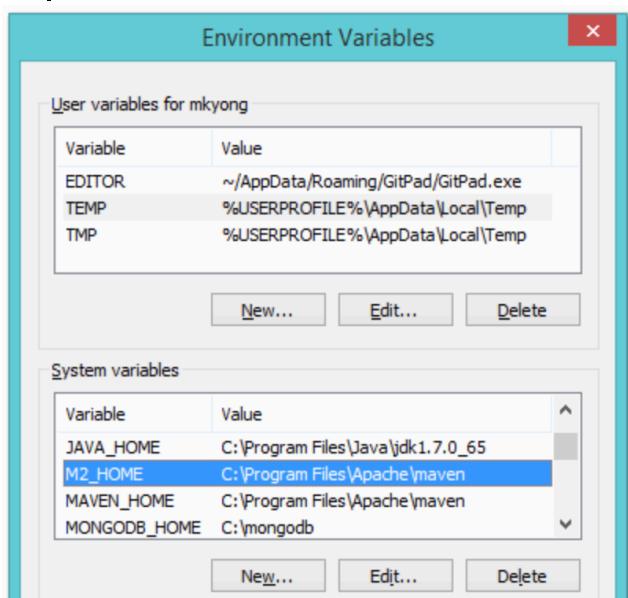


การติดตั้ง

MAVEN

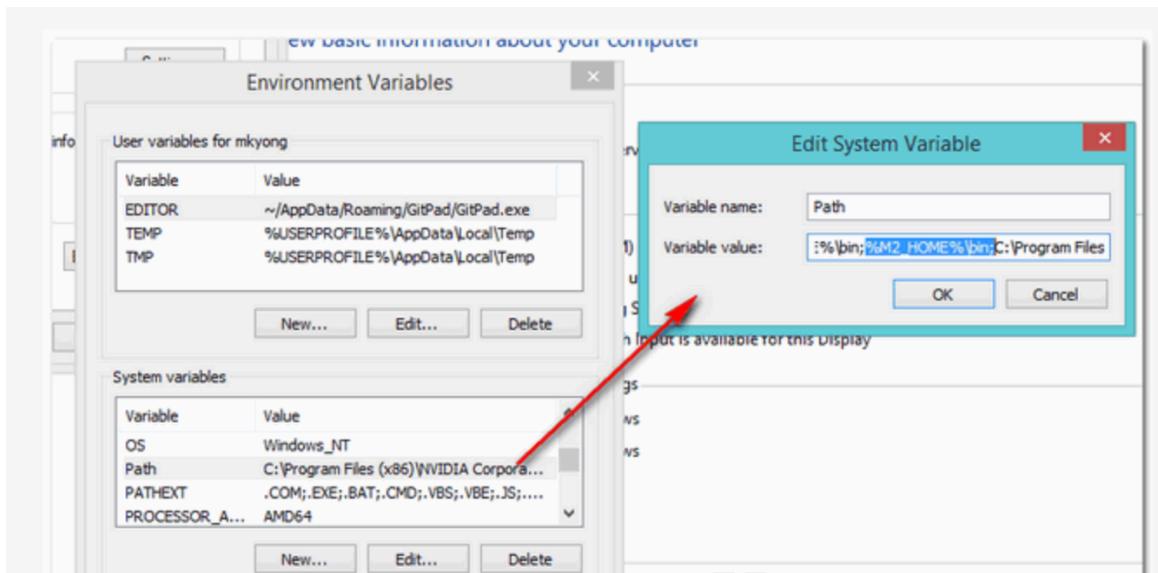
การติดตั้ง maven

- download maven ตาม url ดังนี้
 - <https://maven.apache.org/download.cgi>
 - example :apache-maven-3.2.2-bin.zip
- หลังจาก download แตก zip file ให้ที่เครื่อง เช่น c:/maven
- นำ path ของ maven ไปเพิ่มใน environment variable



การติดตั้ง maven

- เพิ่ม PATH เข้าไปใน environment ดังภาพ



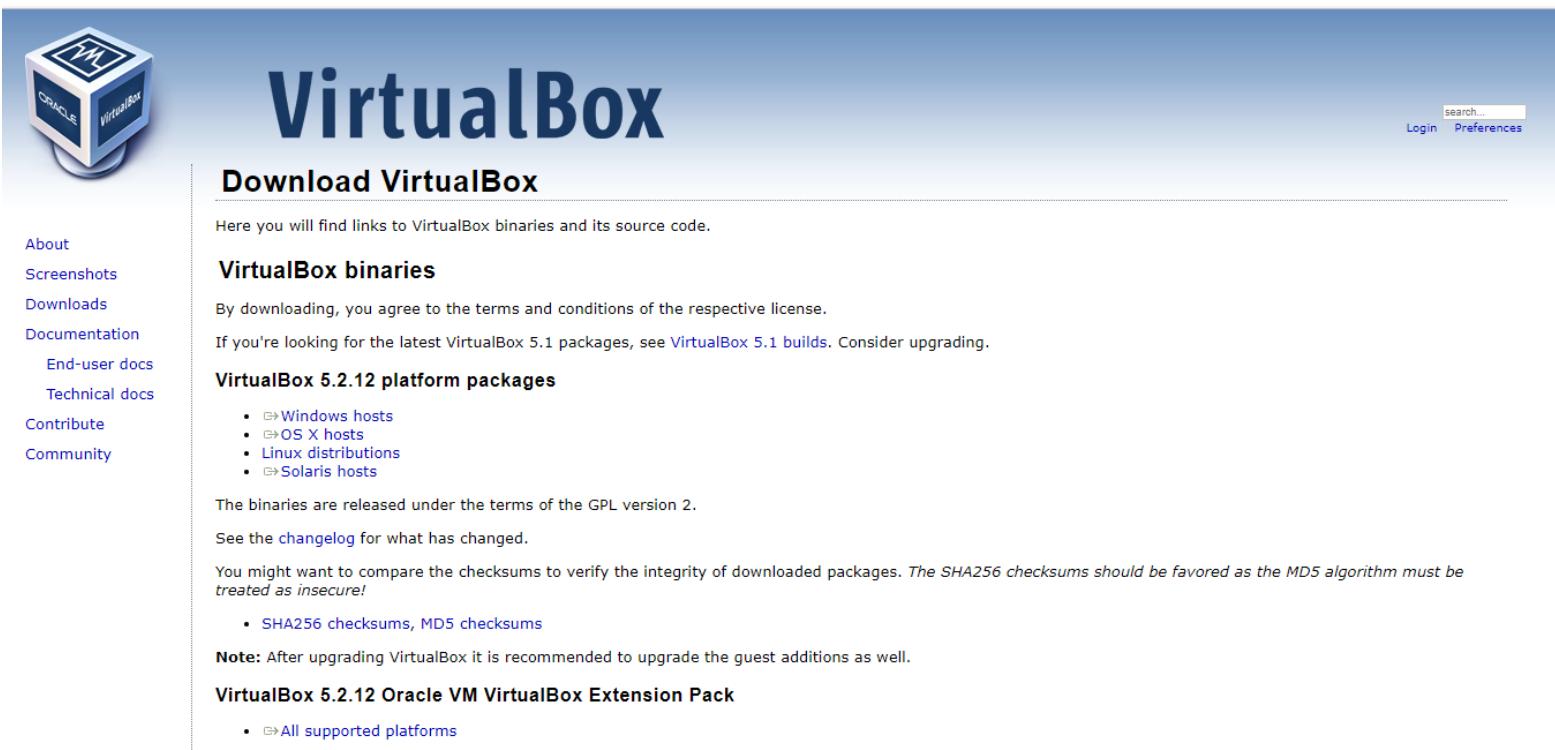
- หลังจากติดตั้งเสร็จแล้วให้ทำการทดสอบการติดตั้งโดยใช้คำสั่งดังนี้
 - mvn -version

การติดตั้ง

VIRTUAL BOX

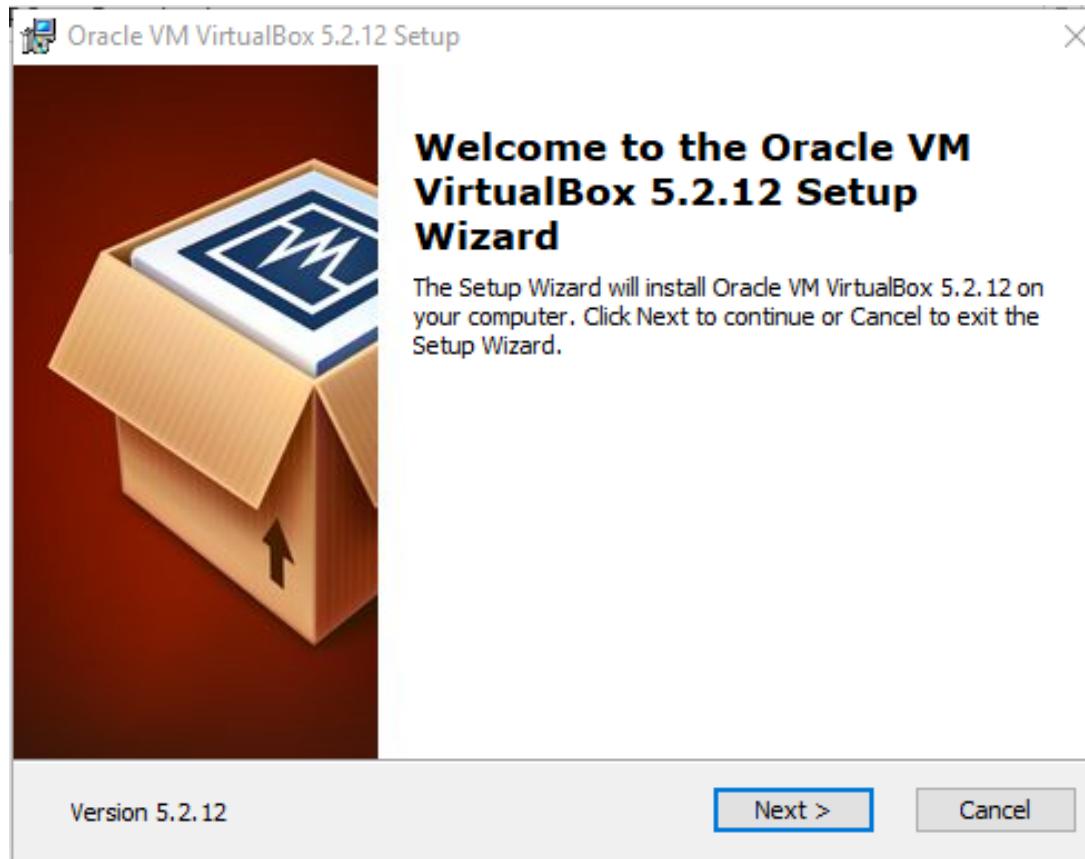
การติดตั้ง Virtual Box

- <https://www.virtualbox.org/wiki/Downloads>

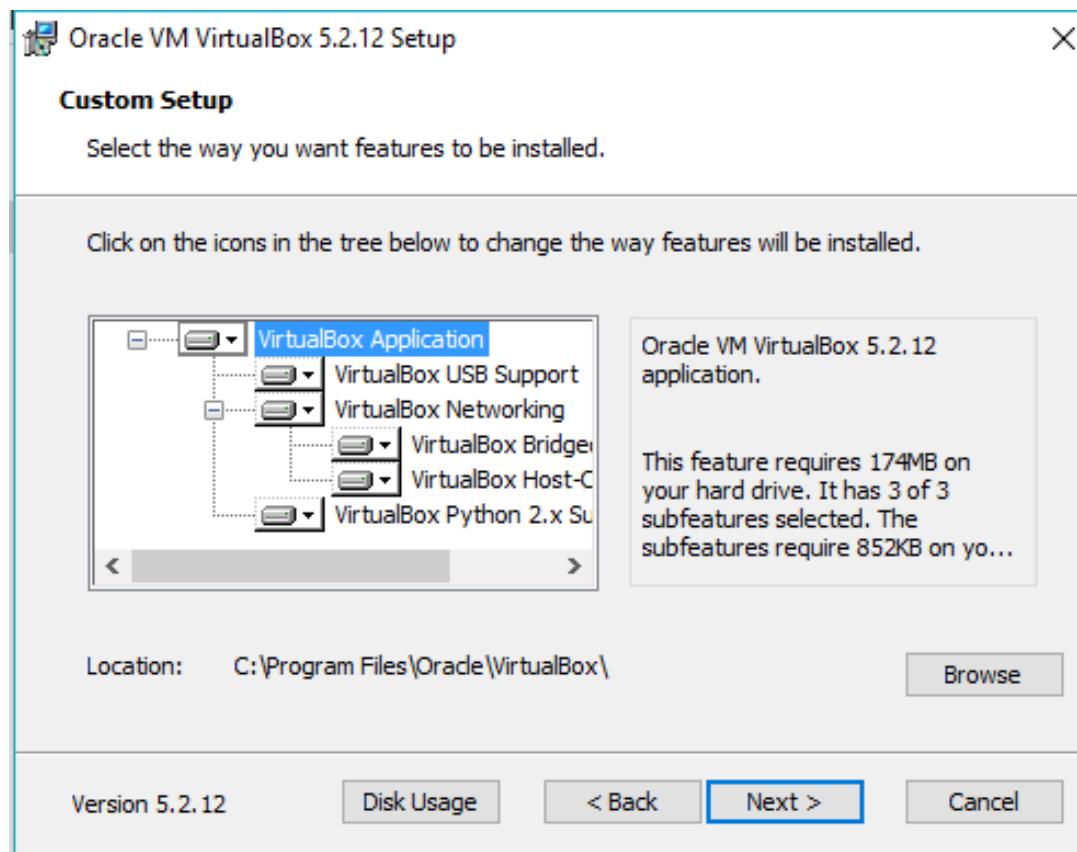


The screenshot shows the official VirtualBox download page. At the top left is the Oracle VM VirtualBox logo, which is a blue cube with the letters 'V' and 'M' on it, and 'ORACLE' and 'VirtualBox' written on its sides. To the right of the logo is the large title 'VirtualBox'. Below the title is a sub-header 'Download VirtualBox'. A horizontal dotted line separates this from the main content area. On the left side of the content area is a sidebar with links: 'About', 'Screenshots', 'Downloads', 'Documentation', 'End-user docs', 'Technical docs', 'Contribute', and 'Community'. The 'Downloads' link is highlighted in blue. The main content area starts with a paragraph: 'Here you will find links to VirtualBox binaries and its source code.' Below this is a section titled 'VirtualBox binaries' with the sub-instruction: 'By downloading, you agree to the terms and conditions of the respective license.' Another note below it says: 'If you're looking for the latest VirtualBox 5.1 packages, see [VirtualBox 5.1 builds](#). Consider upgrading.' Underneath is a section for 'VirtualBox 5.2.12 platform packages' with a bulleted list: 'Windows hosts', 'OS X hosts', 'Linux distributions', and 'Solaris hosts'. Below this is a note: 'The binaries are released under the terms of the GPL version 2.' Further down is a link to the 'changelog'. A note about checksums follows: 'You might want to compare the checksums to verify the integrity of downloaded packages. *The SHA256 checksums should be favored as the MD5 algorithm must be treated as insecure!*' There is a bullet point: 'SHA256 checksums, MD5 checksums'. A note about guest additions follows: 'Note: After upgrading VirtualBox it is recommended to upgrade the guest additions as well.' Finally, there is a section for 'VirtualBox 5.2.12 Oracle VM VirtualBox Extension Pack' with a bullet point: 'All supported platforms'.

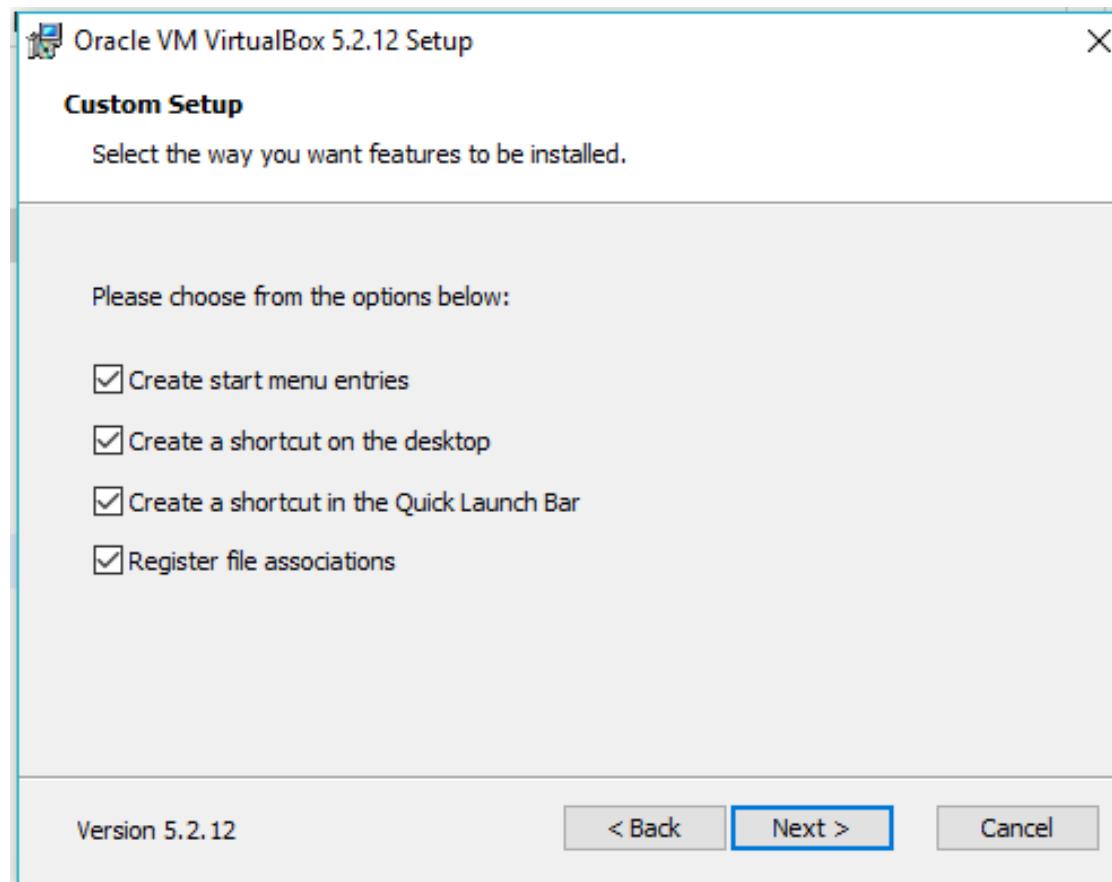
การติดตั้ง Virtual Box



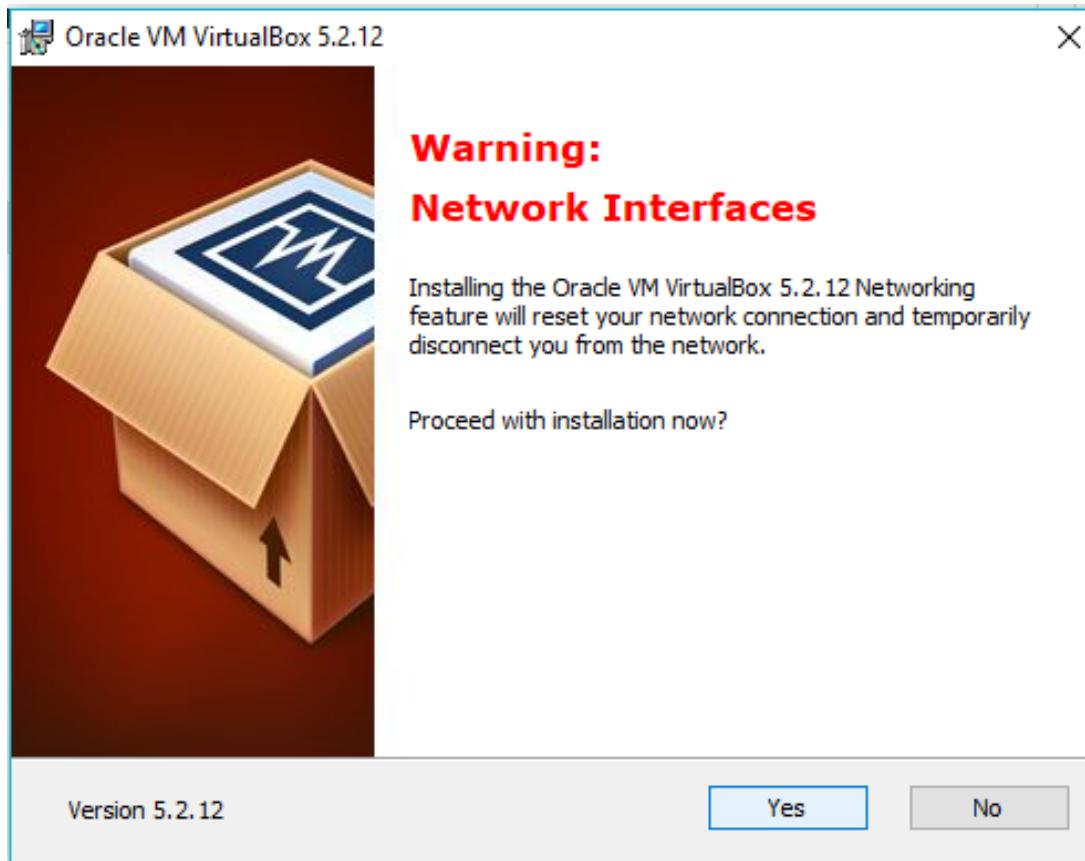
การติดตั้ง Virtual Box



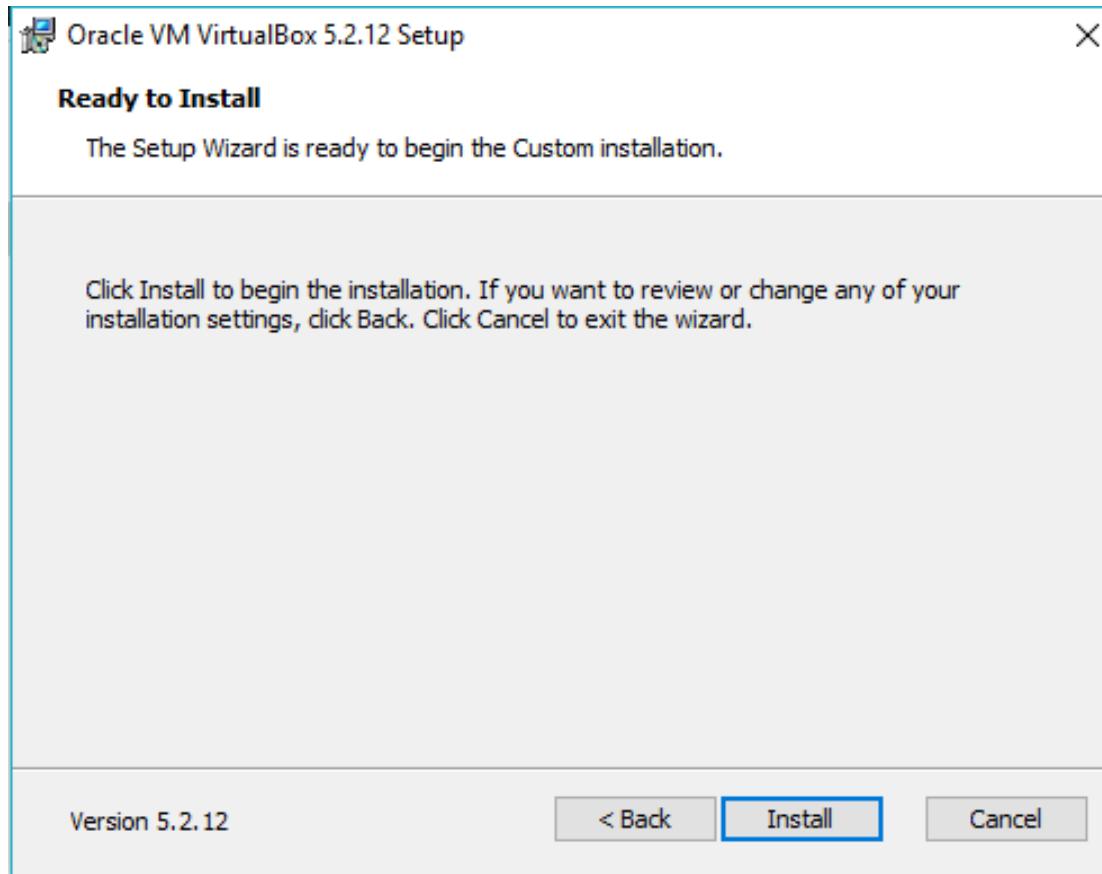
การติดตั้ง Virtual Box



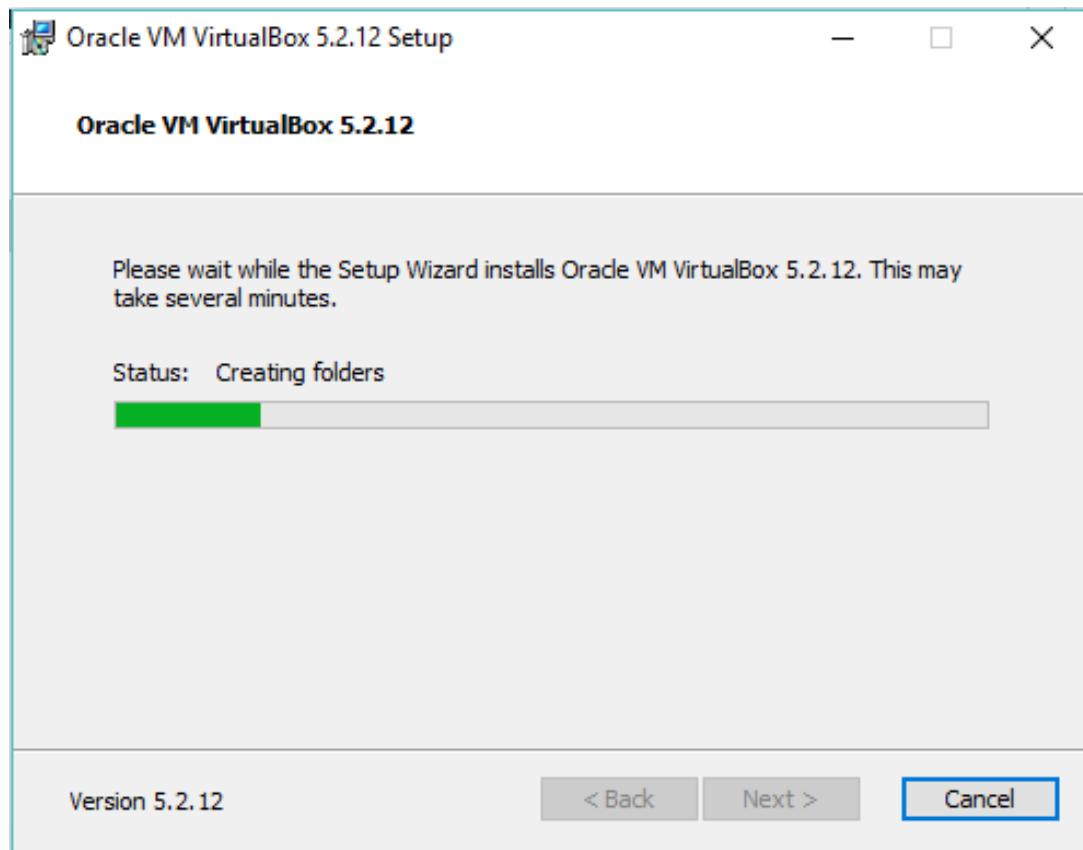
การติดตั้ง Virtual Box



การติดตั้ง Virtual Box



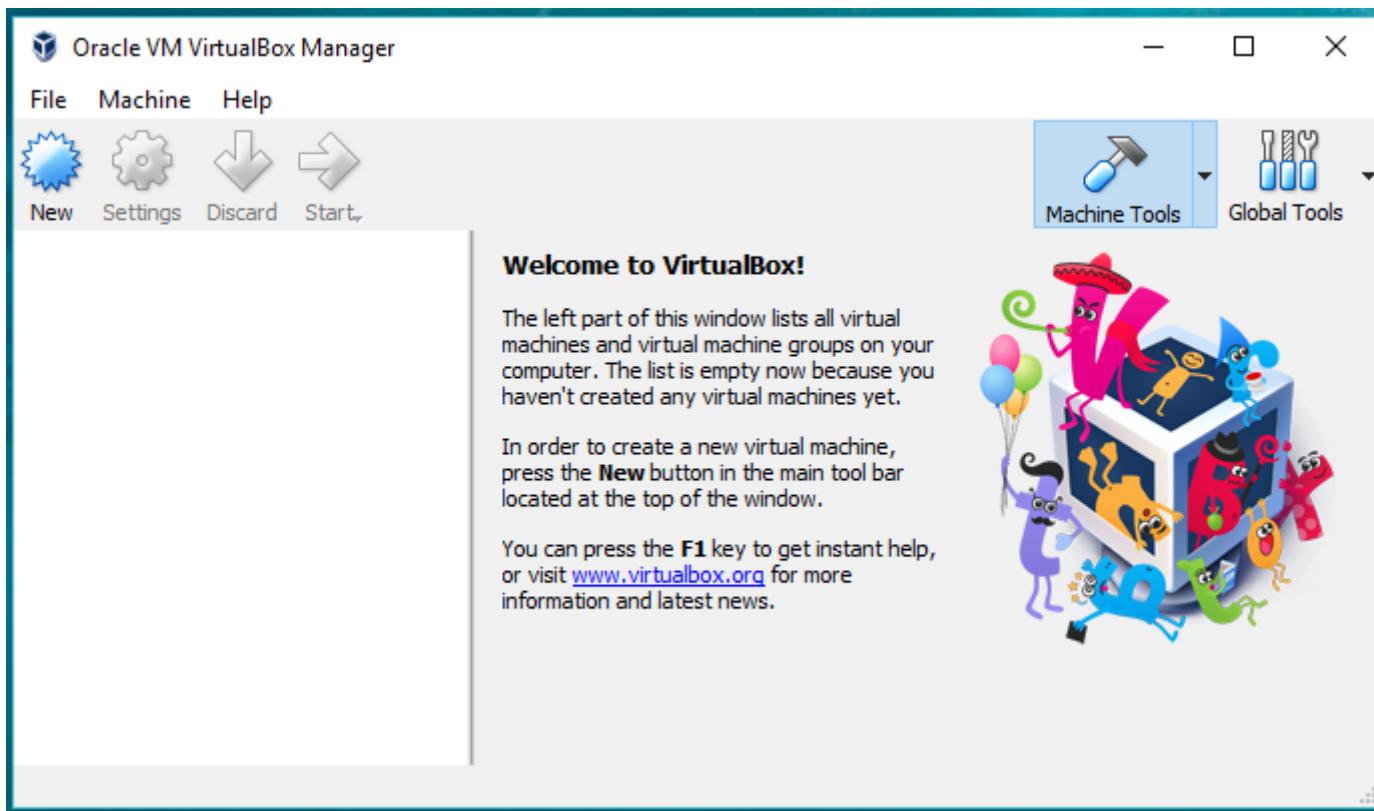
การติดตั้ง Virtual Box



การติดตั้ง Virtual Box



การติดตั้ง Virtual Box



การติดตั้ง

DOCKER TOOLBOX

การติดตั้ง Docker Toolbox

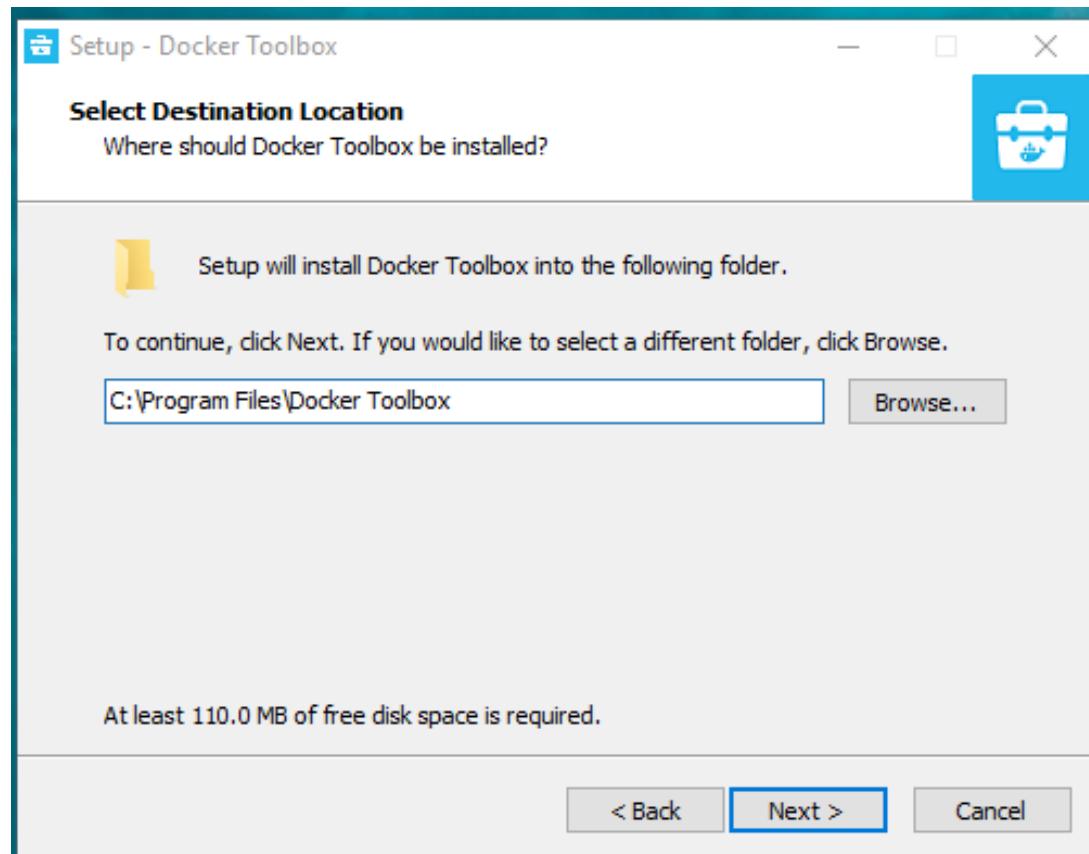
- <https://docs.docker.com/toolbox/overview/#ready-to-get-started>

The screenshot shows the Docker Documentation website. The top navigation bar includes links for 'Search the docs', 'Guides', 'Product manuals', 'Glossary', 'Reference', and 'Samples'. On the right, it shows 'Docker v18.03 (current)'. The left sidebar has a 'Get Docker' section with links for 'Overview of Docker editions', 'Docker CE', 'About Docker CE', 'Cloud', 'Linux', 'MacOS', 'Microsoft Windows', 'Docker CE Edge releases', 'Docker Toolbox (legacy)', 'Toolbox overview', 'Install Toolbox on Mac', 'Install Toolbox on Windows', 'Kitematic', and 'Troubleshooting'. The main content area features a heading 'Ready to get started?'. Step 1: 'Get the latest Toolbox installer for your platform:' shows two buttons: 'Toolbox for Mac' and 'Toolbox for Windows', both highlighted with a red border. Step 2: 'Choose the install instructions for your platform, and follow the steps:' lists 'Install Docker Toolbox on macOS' and 'Install Docker Toolbox for Windows'. Below this is a 'Next steps' section with links: 'Try the Get started tutorial.', 'Dig in deeper with more tutorials and examples on building images, running containers, networking, managing data, and storing images on Docker Hub.', 'Learn about Kitematic', and 'Learn about Docker Machine'.

การติดตั้ง Docker Toolbox

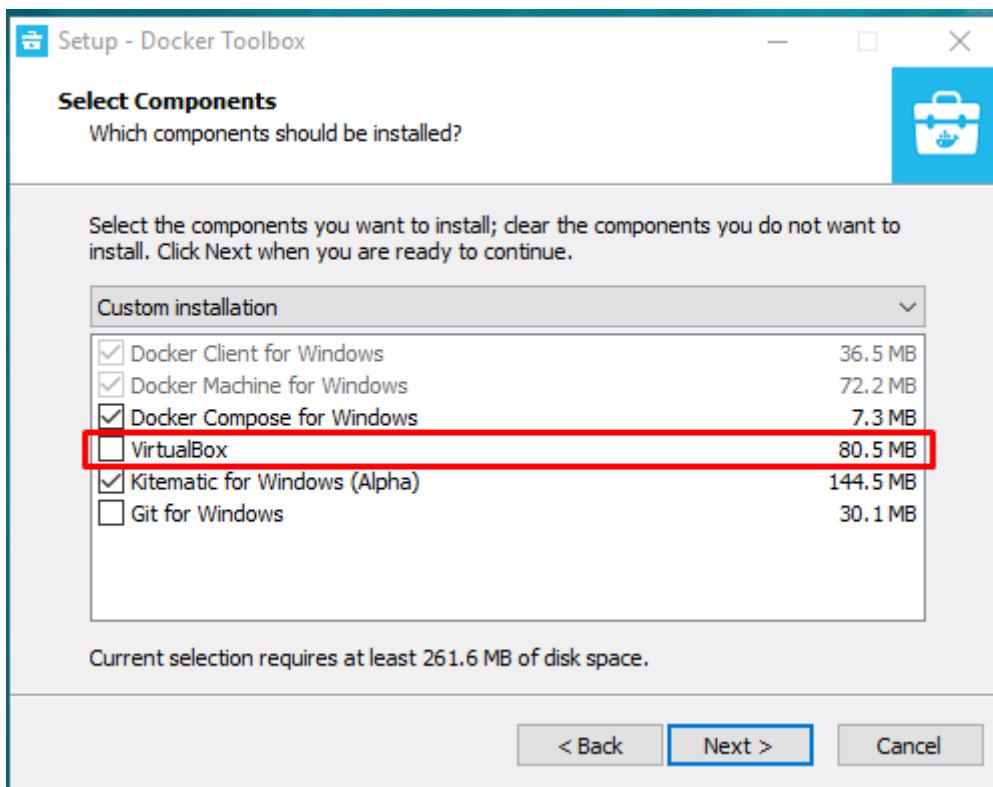


การติดตั้ง Docker Toolbox

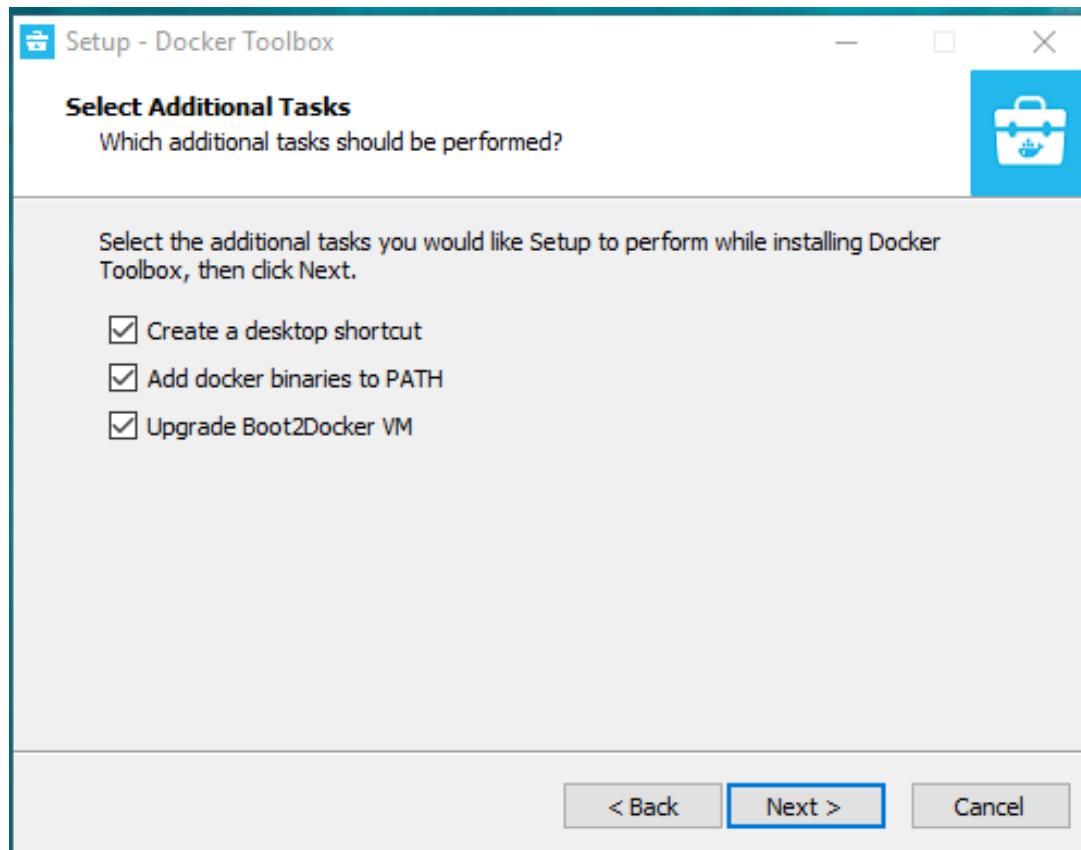


การติดตั้ง Docker Toolbox

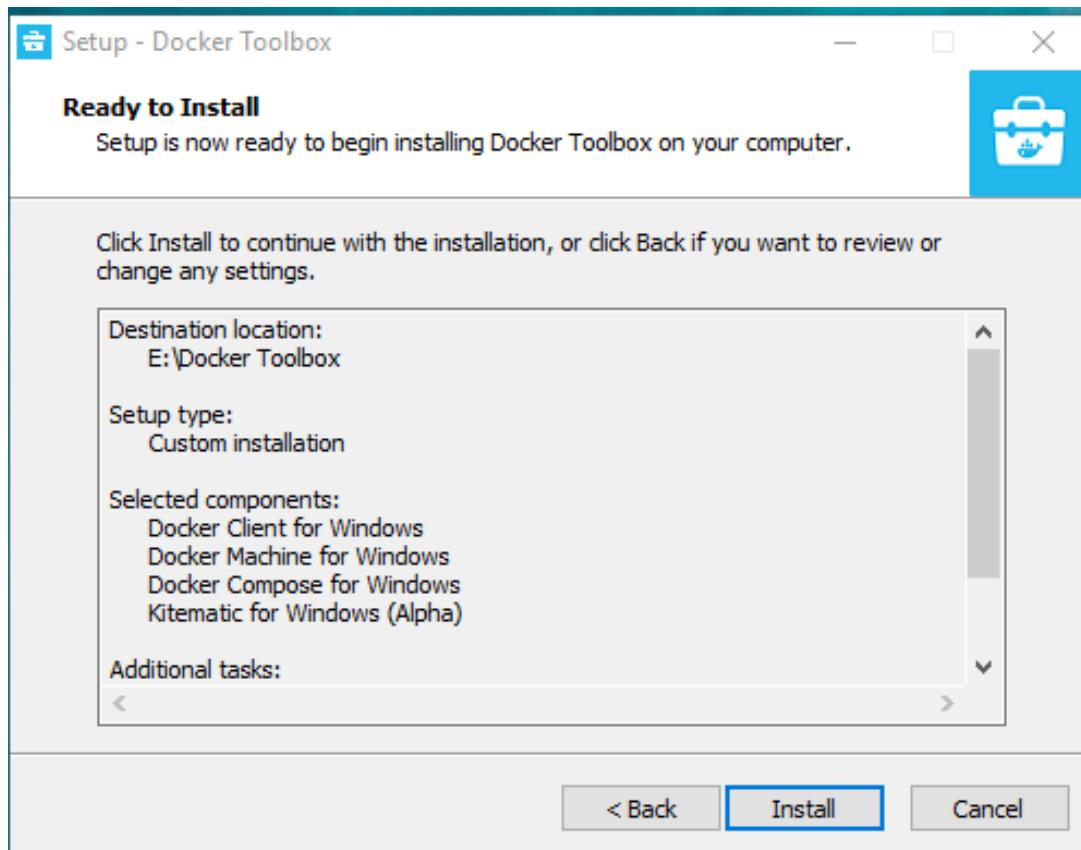
- เอาเครื่องหมายถูกหน้า VirtualBox ออก



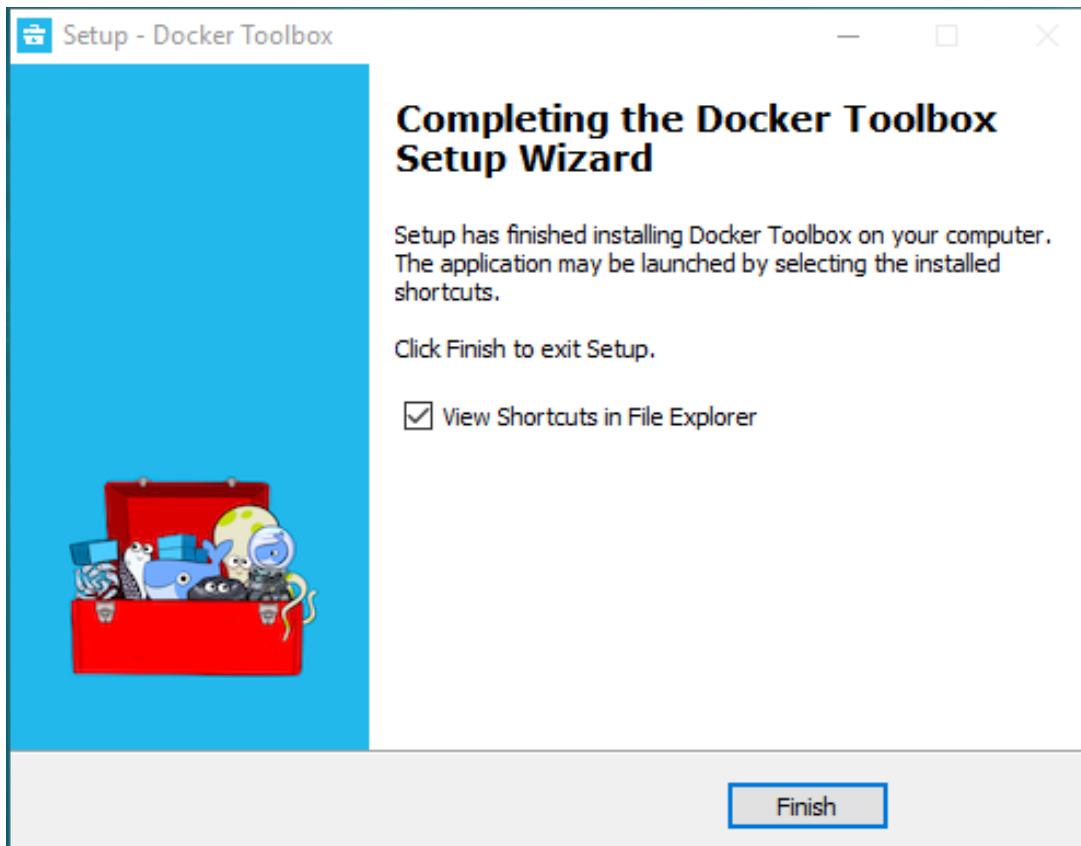
การติดตั้ง Docker Toolbox



การติดตั้ง Docker Toolbox



การติดตั้ง Docker Toolbox



Software project management with

MAVEN

Getting started with Maven

- According to the docs, Maven is a “software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project’s build, reporting and documentation from a central piece of information.” However, most normal people refer to it as a build tool. Its job is to build the project, convert the source code into a binary artifact, package the resources into it, and if needed automatically download and use necessary dependencies.
- The complexity of using Maven comes from the fact that it tries to do multiple things at the same time. First of all it needs to describe the project, its structure, submodules, the necessary build steps and so on. Then it needs a mechanism to provide the information about what other libraries are required for the project to build. Then it has to actually assemble the project, run the tests and perhaps even push it out to your artifact repository.

Create project

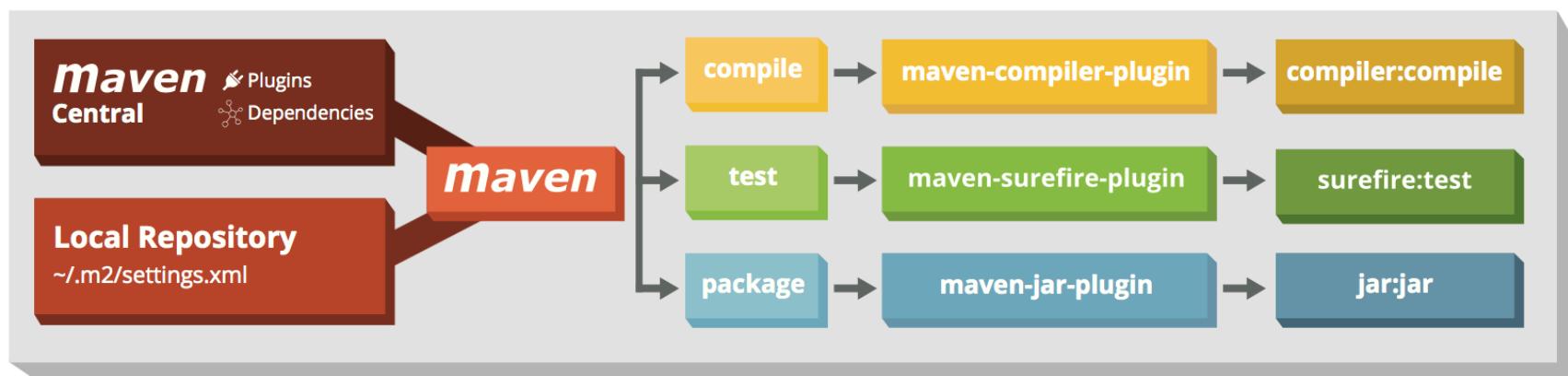
Create Java Project

- mvn archetype:generate -DgroupId=org.yourcompany.project - DartifactId=application

Create web project

- mvn archetype:generate -DgroupId=org.yourcompany.project - DartifactId=application -DarchetypeArtifactId=maven-archetype-webapp

Maven architecture



Maven command

- clean— delete target directory
- validate— validate, if the project is correct
- compile— compile source code, classes stored in target/classes
- test— run tests
- package— take the compiled code and package it in its distributable format, e.g. JAR, WAR
- verify— run any checks to verify the package is valid and meets quality criteria
- install— install the package into the local repository
- deploy— copies the final package to the remote repository

POM

- A Project Object Model or POM is the fundamental unit of work in Maven.
- It is an XML file that contains information about the project and configuration details used by Maven to build the project.
- It contains default values for most projects.
- Examples for this is the build directory, which is target; the source directory, which is src/main/java; the test source directory, which is src/test/java; and so on.
- When executing a task or goal, Maven looks for the POM in the current directory.
- It reads the POM, gets the needed configuration information, then executes the goal.
- Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on.
- Other information such as the project version, description, developers, mailing lists and such can also be specified.

Introduction

SPRING BOOT

Introduction

- Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
- We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need very little Spring configuration.

Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

New spring boot project (WEB)

- goto url <https://start.spring.io/>

Generate a with and Spring Boot

Project Metadata

Artifact coordinates

Group

Artifact

Name

Description

Package Name

Packaging

Java Version

Too many options? [Switch back to the simple version.](#)



Dependencies

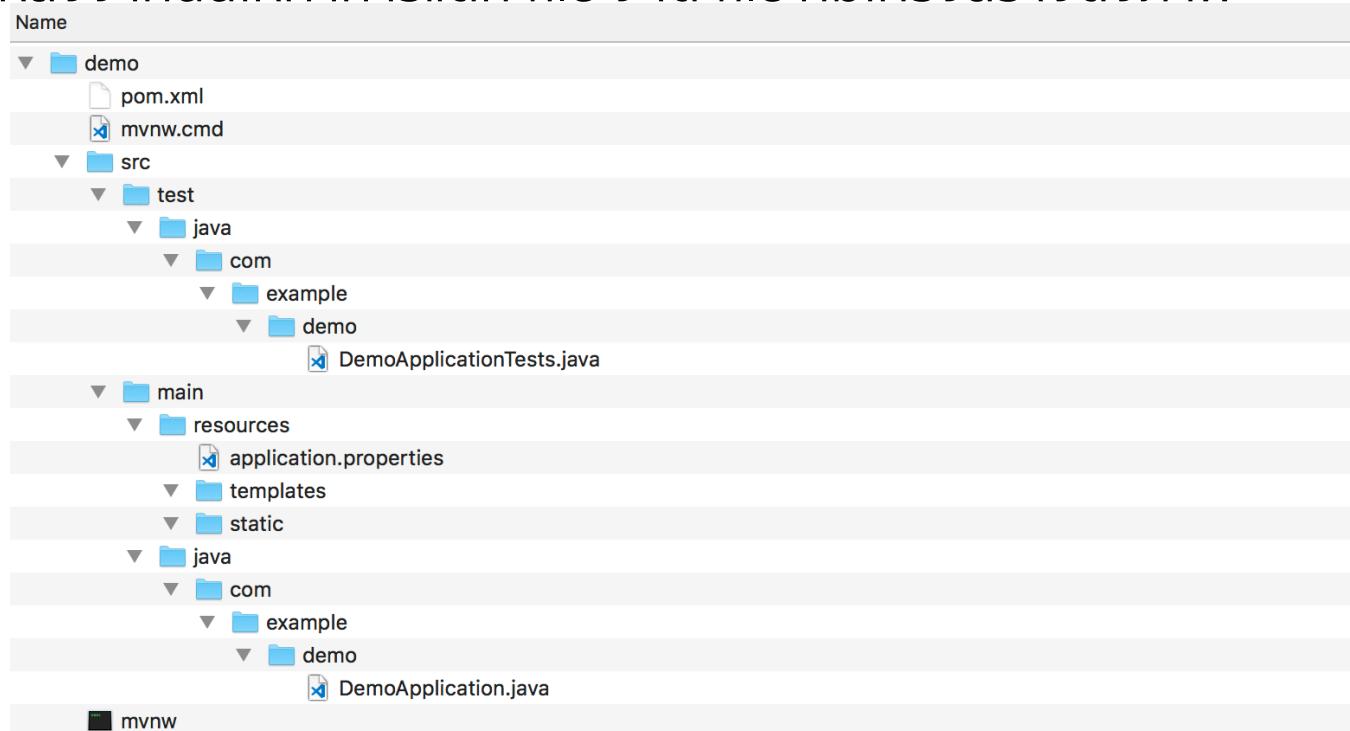
Add Spring Boot Starters and dependencies to your application

Search for dependencies

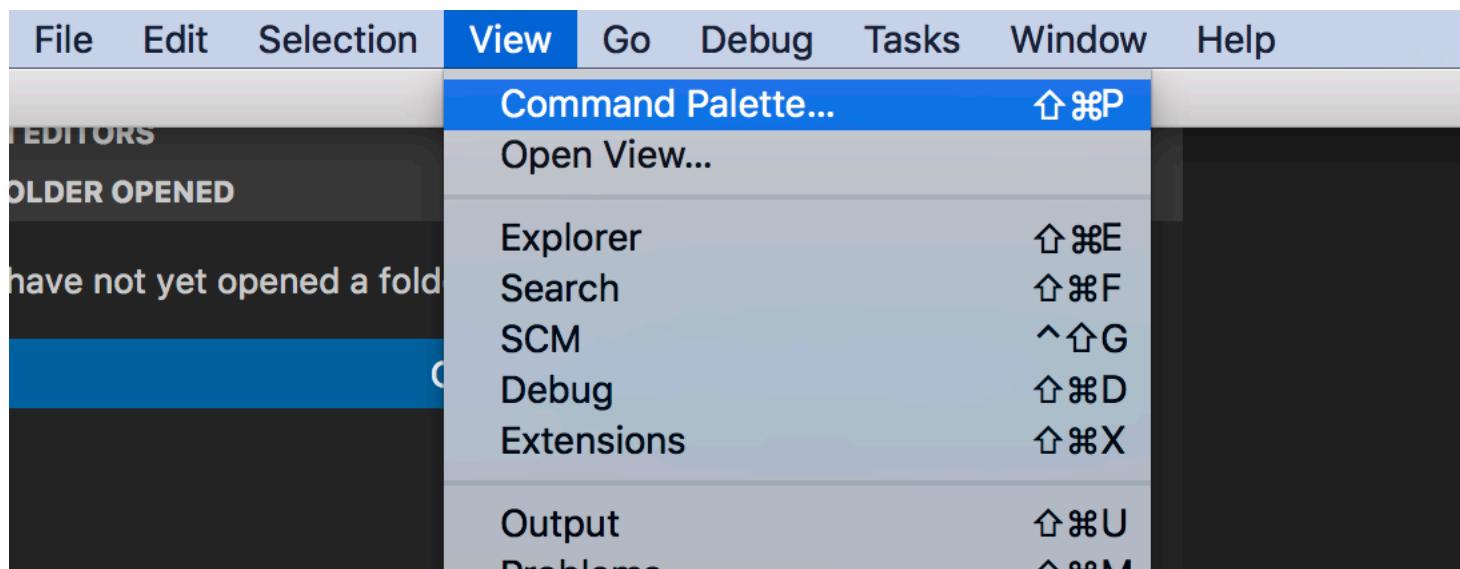
Selected Dependencies

New spring boot project (WEB)

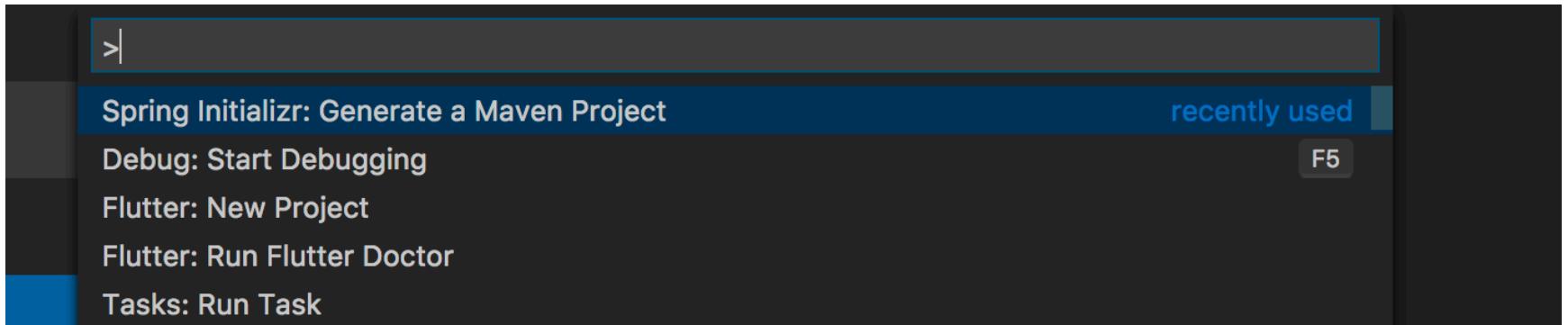
- หลังจากกดปุ่ม Generate Project ตัว web จะทำการสร้าง file นามสกุล zip มาให้ตามชื่อ artifact ที่ได้ระบุไว้ เช่น demo.zip
- หลังจากนั้นให้ทำการแตก file จะได้ file ที่มีโครงสร้างดังภาพ



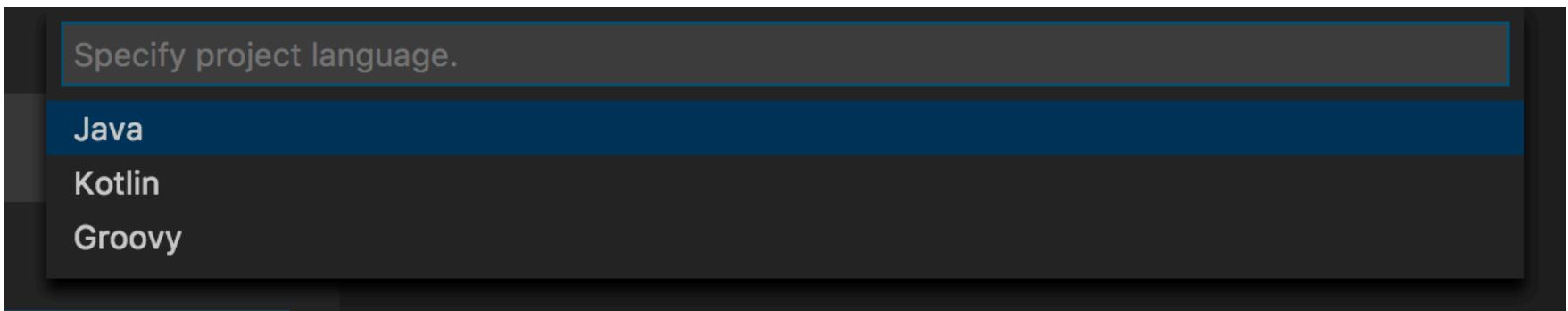
New spring boot project (VS Code)



New spring boot project (VS Code)



เลือก Spring Initializr: Generate a Maven Project



เลือก Java

New spring boot project (VS Code)

com.example

Input Group Id for your project. (Step 1/4) (Press 'Enter' to confirm or 'Escape' to cancel)

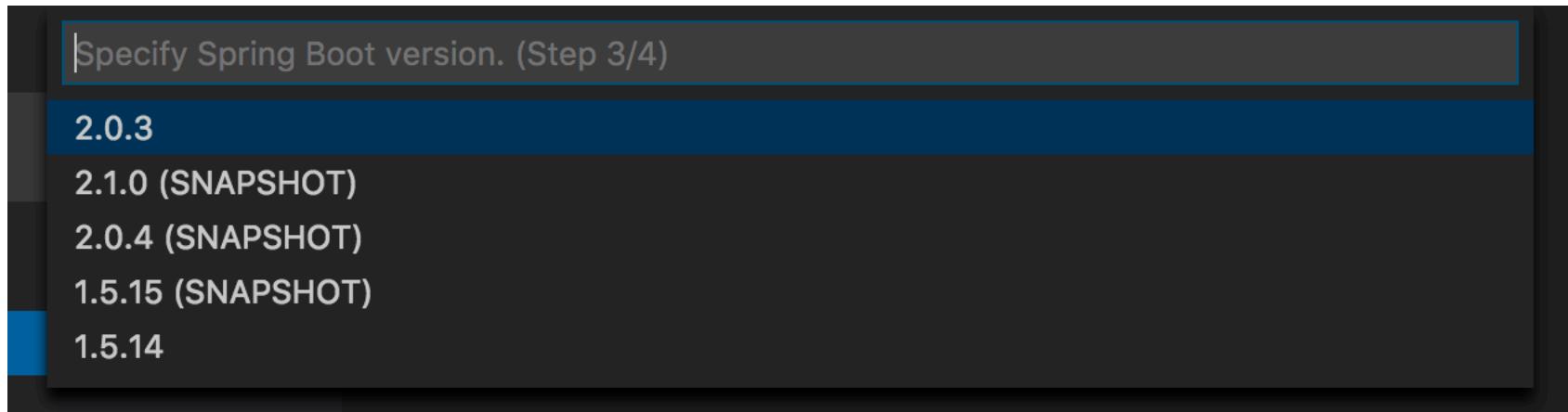
ຮະບຸ Group Id ຂອງ project ວັນນີ້ຈະກລາຍເປັນຫຼື package

demo

Input Artifact Id for your project. (Step 2/4) (Press 'Enter' to confirm or 'Escape' to cancel)

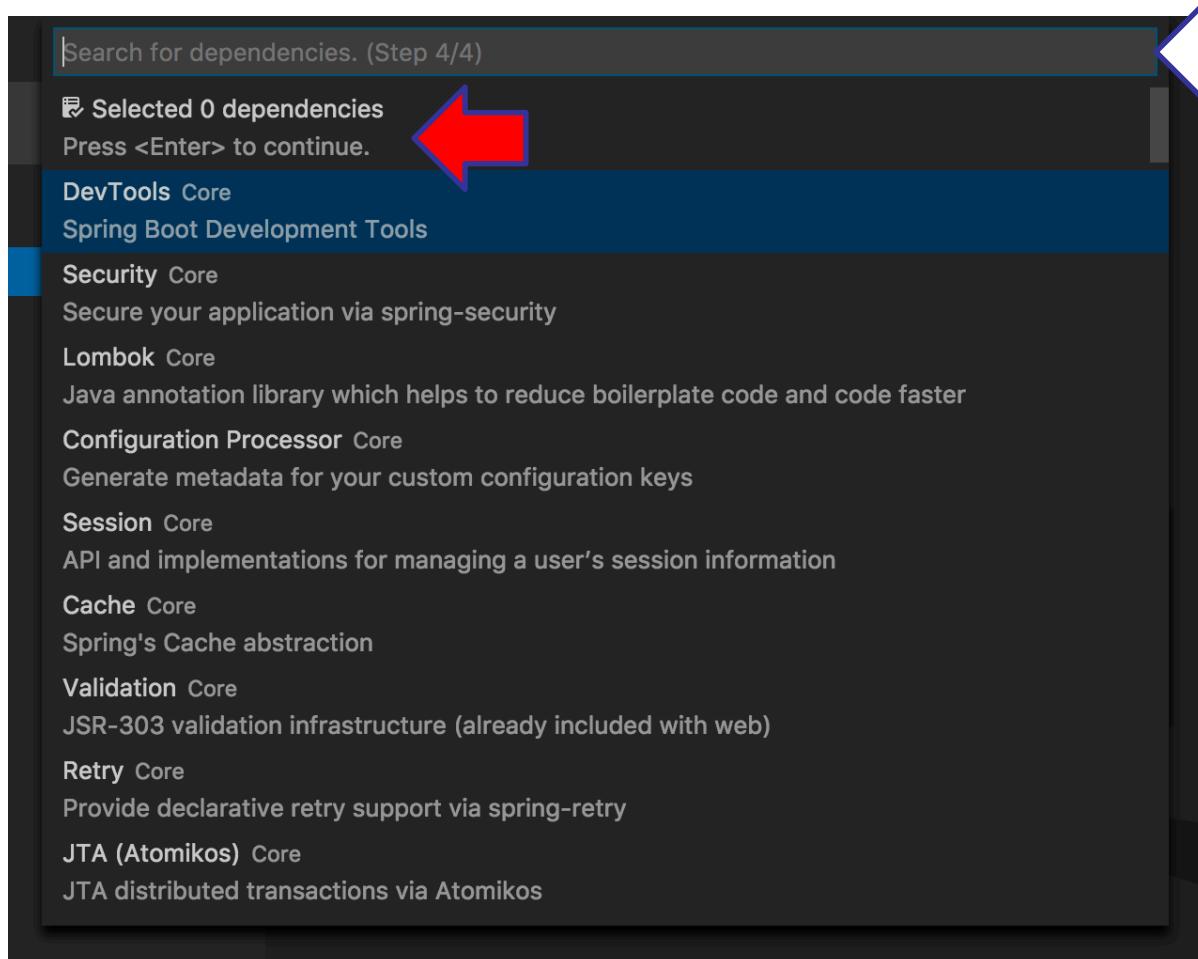
ຮະບຸ Artifact Id ຂອງ project ວັນນີ້ຈະກລາຍເປັນຫຼື project

New spring boot project (VS Code)



เลือก version ของ spring boot

New spring boot project (VS Code)

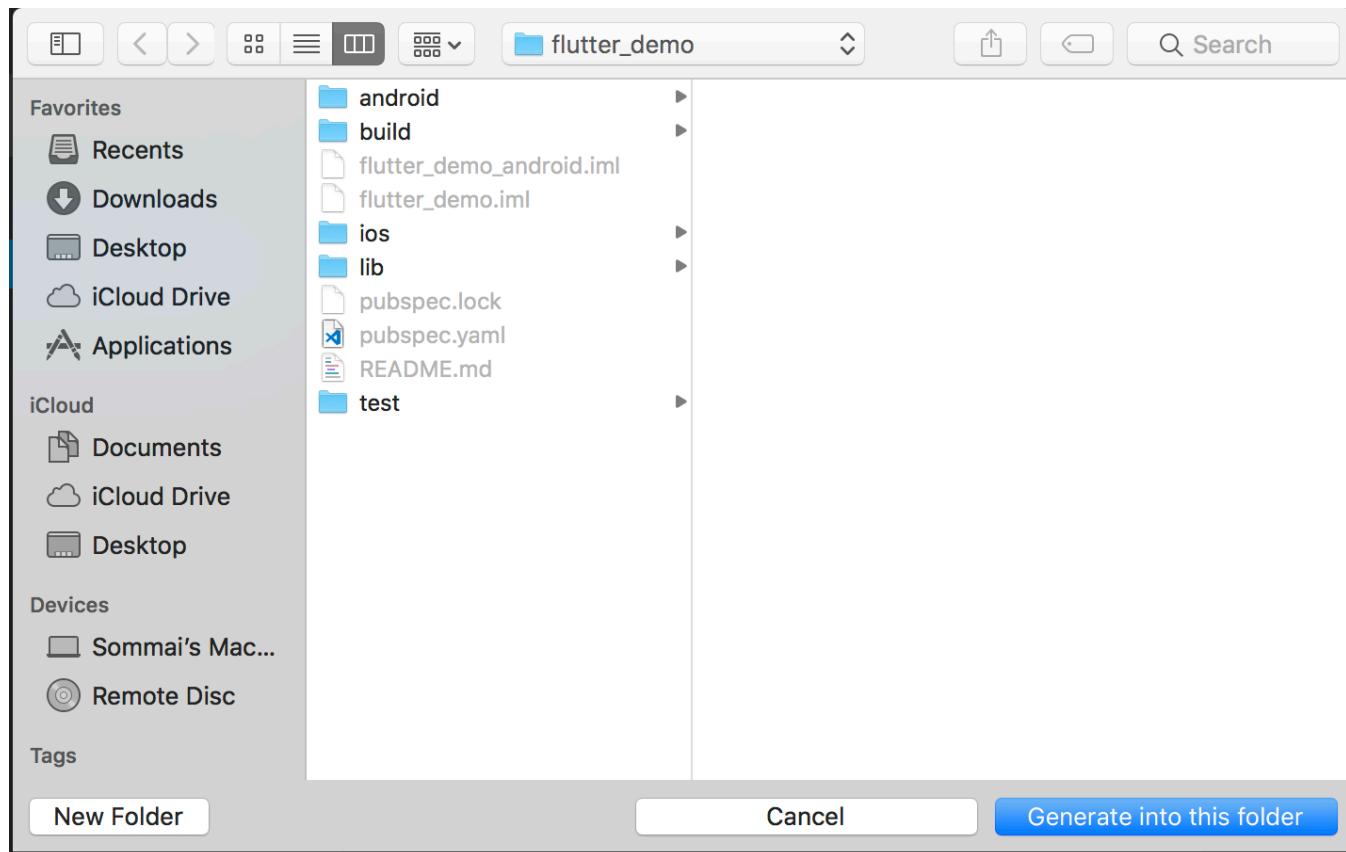


ค้นหา dependencies ที่ต้องการติดตั้ง

หลังจากที่เลือกครบแล้วให้กดตรงเครื่องหมายลักษณะเดิมเพื่อดำเนินการต่อ

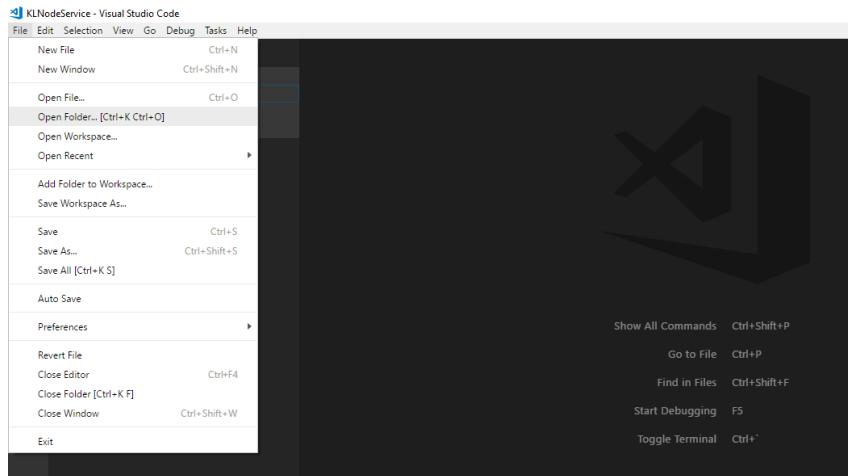
New spring boot project (VS Code)

จะบุ folder ปลายทางที่ต้องการให้สร้าง

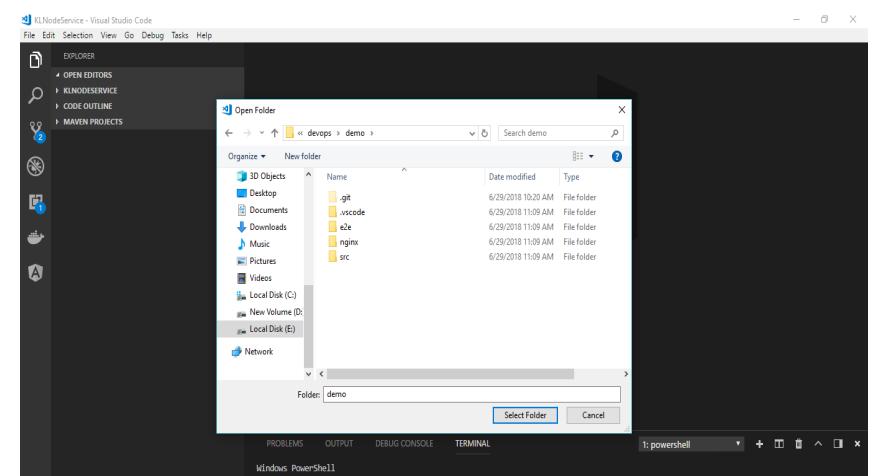


Open project in Visual Studio Code

เลือก menu File > Open Folder



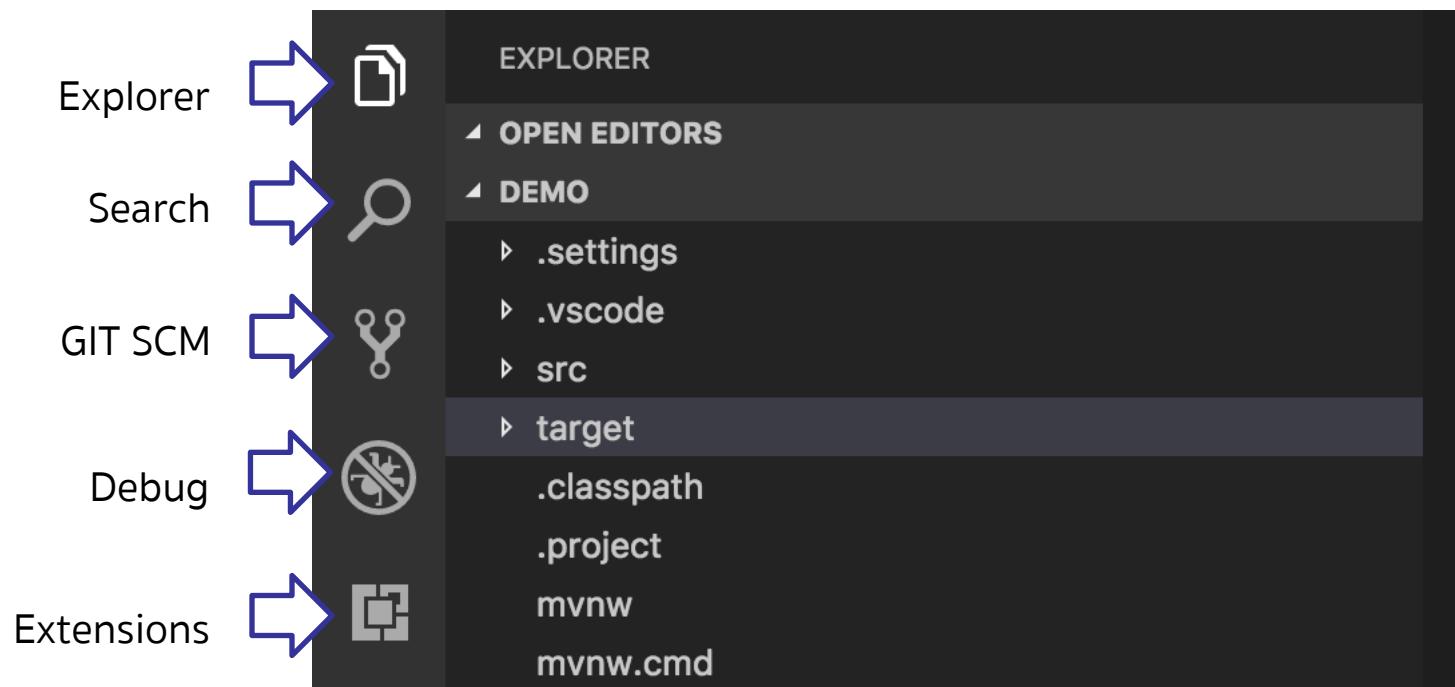
เลือก Folder ที่ต้องการ



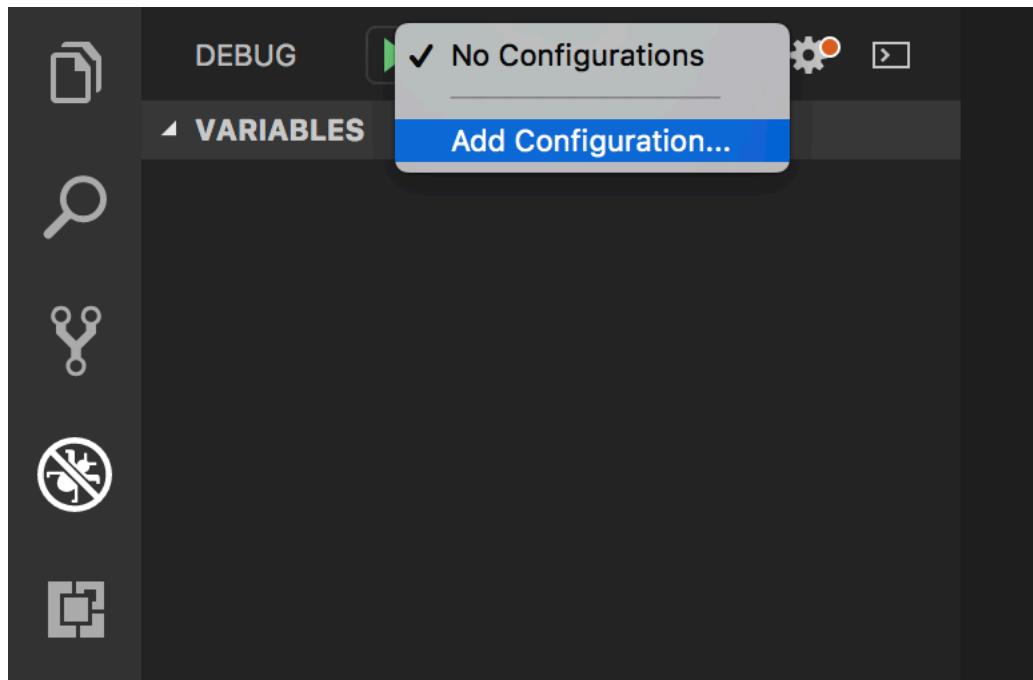
coding spring boot with

VISUAL STUDIO CODE

Overview



Debug



เลือก Add Configuration

Debug

Select Environment

- Chrome
- Docker
- Java
- Node.js
- React Native
- More...

เลือก Java

Debug

- จะมี Debug อยู่ 2 mode ที่ vs code เตรียมไว้ให้คือ
 - launch เป็น Debug จาก class main
 - attach เป็นการ remote debug โดย server ต้อง start แบบ debug mode ด้วยคำสั่งดังนี้
 - mvn spring-boot:run -Dspring-boot.run.jvmArguments="-Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005"
- mode attach ต้องแก้ port เป็น 5005 ดังตัวอย่าง

```
,  
{  
    "type": "java",  
    "name": "Debug (Attach)",  
    "request": "attach",  
    "hostName": "localhost",  
    "port": 5005  
}
```

ขั้นตอนการ Debug

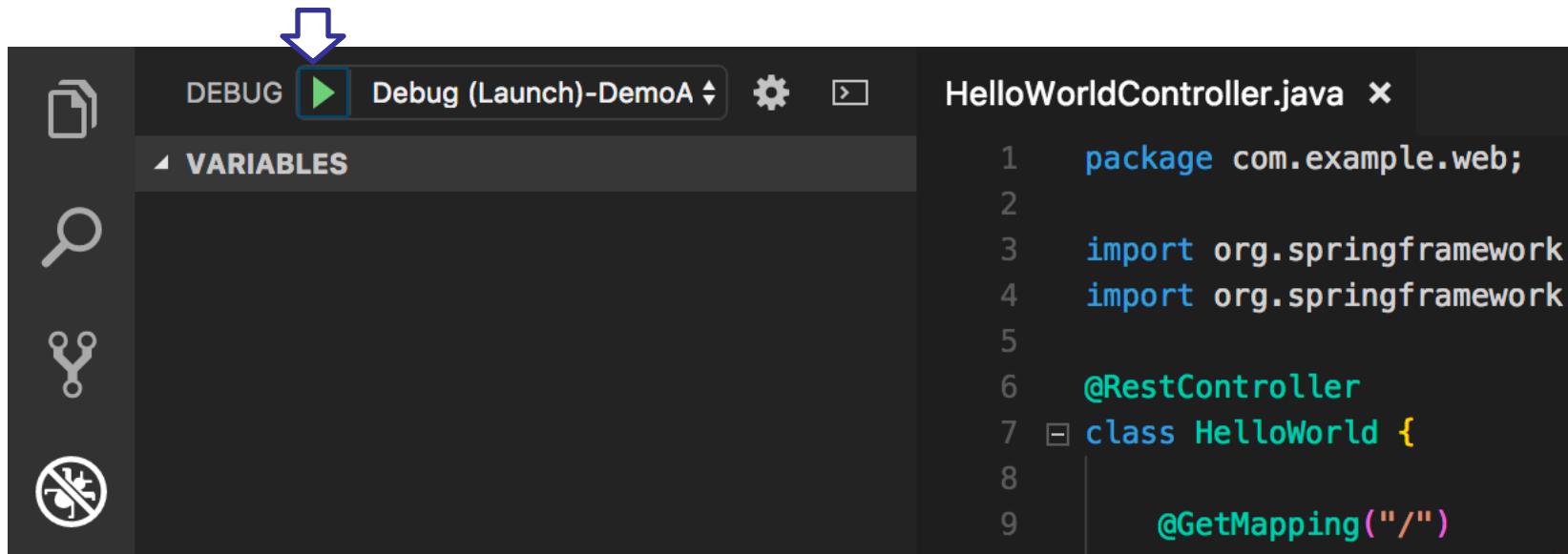


A screenshot of a Java code editor showing a snippet of Java code. The code defines a REST controller named `HelloWorld` with a single endpoint that returns "Hello World". A red circular breakpoint icon is placed on the line before the `String msg = "Hello World";` statement. The code editor has a dark theme with syntax highlighting. A small tab labeled "ava" is visible on the left.

```
5
6     @RestController
7     class HelloWorld {
8
9         @GetMapping("/")
10        public String hello(){
11            String msg = "Hello World";
12            return msg;
13        }
14    }
```

สร้าง breakpoint ใน code ตรงส่วนที่ต้องการ debug โดยกดตรงหน้าตัวเลขหน้าบรรทัด

ขั้นตอนการ Debug



```
1 package com.example.web;
2
3 import org.springframework.
4 import org.springframework.
5
6 @RestController
7 class HelloWorld {
8
9     @GetMapping("/")
10 }
```

กดปุ่ม สีเขียวเพื่อเริ่มการ Debug code

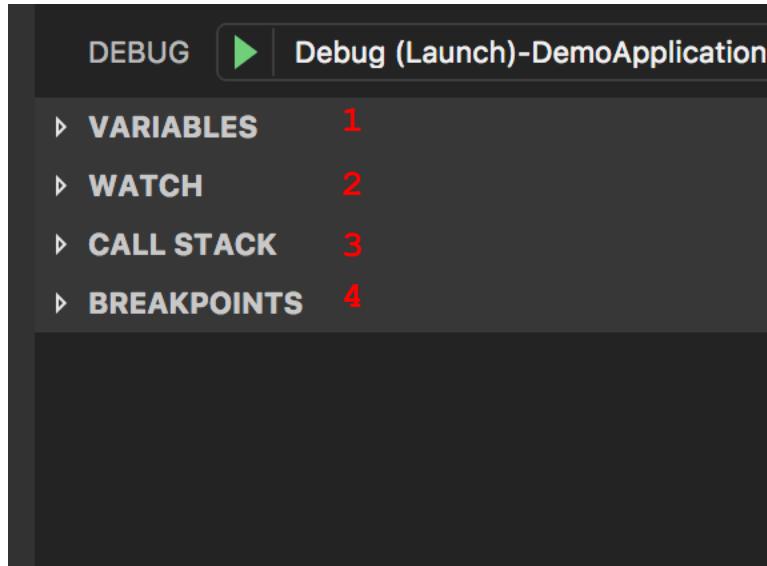
เมื่อ Program ทำงานไปถึงจุดที่สั่ง break point ก็จะหยุดการทำงานเพื่อรอ

ขั้นตอนการ Debug



- 1 = run program ต่อไป
- 2 = ข้ามไปทำคำสั่งต่อไป ภายใน function scope ที่ break point อยู่
- 3 = เข้าไปทำคำสั่งที่อยู่ในภายใน function
- 4 = ย้อนกลับออกจาก function
- 5 = เริ่มการ debug ใหม่
- 6 = หยุดการ debug

ขั้นตอนการ Debug



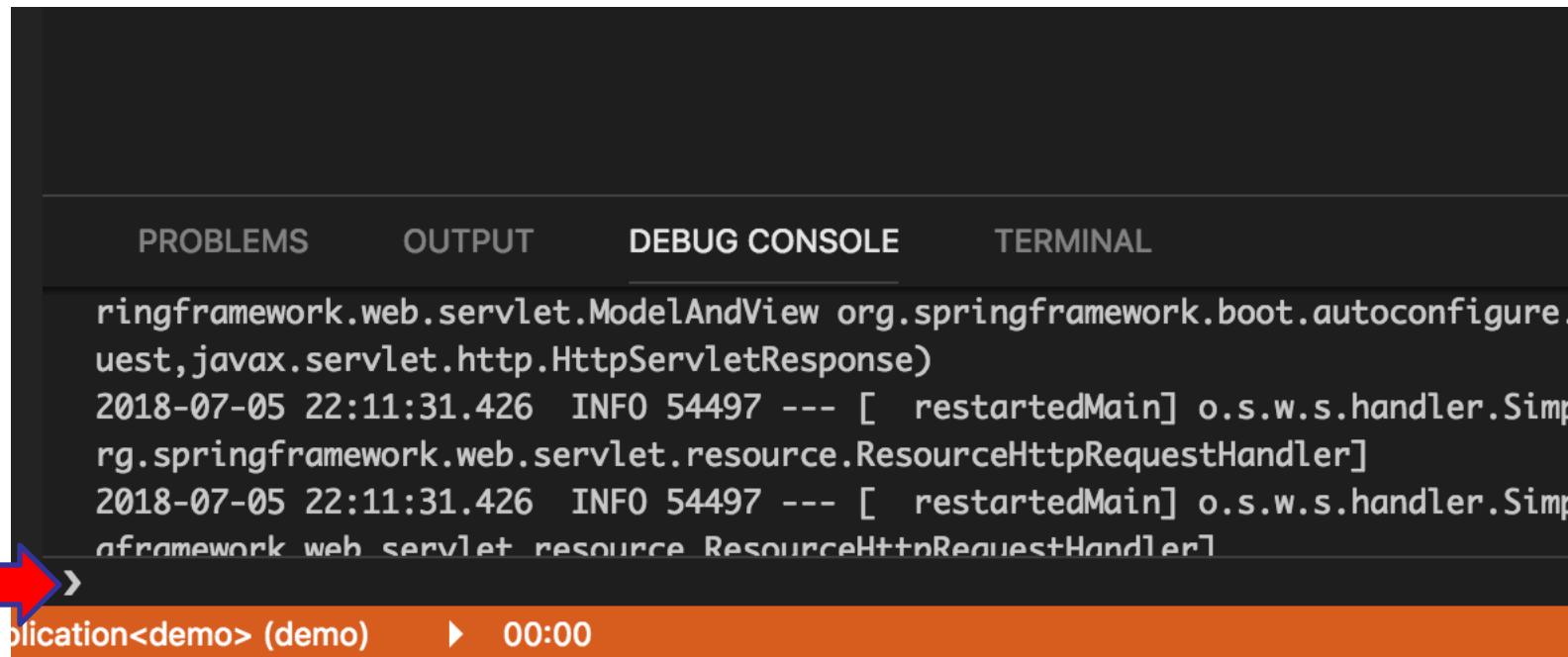
1 = ค่าของตัวแปร

2 = ค่าของตัวแปรที่เลือก

3 = ลำดับขั้นในการเรียก

4 = จุด breakpoint กั้งหมด

ขั้นตอนการ Debug



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

ringframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.
uest, javax.servlet.http.HttpServletResponse)
2018-07-05 22:11:31.426 INFO 54497 --- [ restartedMain] o.s.w.s.handler.Simp
rg.springframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-07-05 22:11:31.426 INFO 54497 --- [ restartedMain] o.s.w.s.handler.Simp
nframework web servlet resource ResourceHttpRequestHandler]

Application<demo> (demo) ▶ 00:00
```

Debug Console คือส่วนที่แสดง log ของการทำงานของโปรแกรม

สามารถพิมพ์คำสั่งเพื่อทดสอบโปรแกรมได้ที่ช่องตรงลูกศรสีแดง

Rest API with

SPRING REST

Controller [@RestController]

```
@RestController
```

```
public class MainController {
```

```
    @RequestMapping("/")
```

```
    public String index(Model model) {
```

```
        return "Hello World";
```

```
}
```

```
}
```

@RequestMapping

```
@RequestMapping("/url")  
  
@RequestMapping(method = RequestMethod.GET, value = "/url")  
  
@RequestMapping(method = RequestMethod.POST)  
public Map<String, Object> save (  
    @RequestBody Map<String, Object> input,  
    @RequestParam (value="name", required=false, defaultValue="World") String name  
) {}  
  
@RequestMapping(method = RequestMethod.GET, value = "/{name}")  
public ArrayList<Map<String, Object>> index(  
    @PathVariable String name  
) { }
```

REST Mapping

@GetMapping ใช้สำหรับ map method get

@PutMapping ใช้สำหรับ map method put

@PostMapping ใช้สำหรับ map method post

@DeleteMapping ใช้สำหรับ map method delete

@ModelAttribute ใช้สำหรับ map submit form model

connect to rdbms with

SPRING JDBC

ຕັດຕັ້ງ library my sql (pom.xml)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_example
```

```
spring.datasource.username=springuser
```

```
spring.datasource.password=ThePassword
```

Example call jdbc with sql statement

```
public class HelloController {  
  
    @Autowired  
    DataSource dataSource;  
  
    @Autowired  
    Simple simple;  
  
    @Value("${spring.datasource.url}")  
    String url;  
  
    @RequestMapping(method = RequestMethod.GET, value = "/{name}")  
    public ArrayList<Map<String, Object>> index(@PathVariable String name) {  
        ArrayList<Map<String, Object>> arr = new ArrayList<>();  
        String v1 = this.simple.methodOne();  
        String rest = this.simple.methodTwo(v1);  
        Map<String, Object> ret = new HashMap<>();  
        ret.put("result", "Hello World zzzzy " + name + rest);  
        arr.add(ret);  
        try{  
            Statement stmt = this.dataSource.getConnection().createStatement();  
            ResultSet res = stmt.executeQuery("select * from ms_customer");  
            while(res.next()){  
                int cCount = res.getMetaData().getColumnCount();  
            }  
        } catch (SQLException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Data repository with

SPRING DATA JPA

application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_example
spring.datasource.username=springuser
spring.datasource.password=ThePassword
spring.datasource.driver=com.mysql.jdbc.Driver
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

dependency [pom.xml]

```
<!-- JPA Data (We are going to use Repositories, Entities, Hibernate, etc...) -->
```

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
<!-- Use MySQL Connector-J -->
```

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
</dependency>
```

Entity

Class Annotations

- @Entity- Required for Hibernate to store/retrieve instances of the class to/from the database. Every field of the class not marked @Transient will be stored in the database (see below for more on @Transient).
- @Transactional- Most often used on CrudRepository interfaces. Specifies that if any data operation within the class throws a runtime exception, all associated database queries should be rolled back (that is, undone). This can prevent data corruption caused by exceptions.
- @Table-Optional. Specifies the name of the table in the given database used to store the objects properties. If omitted, the table name is generated via the default rules.

Entity

Field Annotations

- `@NotNull`- This is not actually a JPA annotation -- it is part of the `javax.validation.constraintspackage` -- but Hibernate enforces it as well. Specifies that the field may not be null.
- `@Column`- Specifies the name of the column in the given table used to store the field. If omitted, the column name is generated via the configured or default naming convention. There are several attributes -- such as `nullable` and `unique`-- that result in database-level restrictions being applied (i.e. in database constraints rather than Hibernate constraints).

Entity

Field Annotations

- **@Id**- The field should be considered an identifier for the class. In particular, the corresponding column in the database should be a primary key. A field with this annotation must one of: a primitive type, a primitive wrapper type (e.g.IntegerorFloat),String,java.util.Date, or a few others (see documentation link below).
- **@GeneratedValue**- Specifies that the field should be generated by Hibernate. This annotation can be configured to use different value generation strategies.
- **@OneToMany**- Specifies a field that is a collection of other persistent objects.

Entity

Field Annotations

- @ManyToOne- The inverse (or "other side") of the @OneToOne annotation. This is used within a class that is "owned" by another class, to annotate a field that points back to the "owner". For example, with Item and Category classes (where Item is contained in a Category, that is, a Category has many Item objects), the Item class can have a category field that points to the object that owns it.
- @OneToOne- Specifies a field that is a other persistent objects.

Entity

Field Annotations

- `@JoinColumn`- Typically used in conjunction with`@OneToOne`, this annotation specifies the column on the "owned" object's table that should be used to identify its owner. In the`Item/Category` example, the value of the`category_id` column specifies the`id` of the`Category` that owns the`Item`.
- `@Transient`- Specifies that Hibernate should not store the field in the database.

Many To one Entity

```
@Entity
@Transactional
@Table( name="sc_user" )
public class User {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    @Column( name="user_name", length=20)
    private String name;

    @NotNull
    @Column( length=15)
    private String email;

    @ManyToOne
    @JoinColumn(name = "profile")
    private Profile profile;

    @Transient
    private String remark;

    public Profile getProfile() {
        return profile;
    }

    public void setProfile(Profile profile) {
        this.profile = profile;
    }

    public User (){
    }
    public User(String name, String email){
        this.name = name;
        this.email = email;
    }
}
```

One to many Entity

```
@Entity
public class Profile {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    private String name;
    private String role;

    @OneToMany( mappedBy="profile", cascade=CascadeType.ALL)
    private Set<User> users;

    public Profile(){

    }

    public Profile(String name, String role){

    }

    public Set<User> getUsers(){
        return this.users;
    }

    public void setUsers(Set<User> users){
        this.users = users;
    }
}
```

One to one Entity

```
@Entity
public class UserDetail {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer id;

    private String phone;

    @OneToOne(fetch=FetchType.LAZY)
    @JoinColumn(name="id")
    private User user;
}
```

Repositories

The central interface in the Spring Data repository abstraction is `isRepository`. It takes the domain class to manage as well as the ID type of the domain class as type arguments. This interface acts primarily as a marker interface to capture the types to work with and to help you to discover interfaces that extend this one. The `CrudRepository` provides sophisticated CRUD functionality for the entity class that is being managed.

CrudRepositoryinterface

```
public interface CrudRepository<T, ID extends Serializable>
    extends Repository<T, ID> {
    <S extends T> S save(S entity);          ①
    Optional<T> findById(ID primaryKey);      ②
    Iterable<T> findAll();                   ③
    long count();                           ④
    void delete(T entity);                 ⑤
    boolean existsById(ID primaryKey);       ⑥
    // ... more functionality omitted.
}
```

CrudRepositoryinterface

- ① Saves the given entity.
- ② Returns the entity identified by the given ID.
- ③ Returns all entities.
- ④ Returns the number of entities.
- ⑤ Deletes the given entity.
- ⑥ Indicates whether an entity with the given ID exists.

Query Creation

The query builder mechanism built into Spring Data repository infrastructure is useful for building constraining queries over entities of the repository. The mechanism strips the prefixes `find...By`, `read...`, `By`, `query...By`, `count...By`, and `get...By` from the method and starts parsing the rest of it. The introducing clause can contain further expressions, such as `aDistinct` to set a distinct flag on the query to be created. However, the first `By` acts as delimiter to indicate the start of the actual criteria. At a very basic level, you can define conditions on entity properties and concatenate them with `And` and `Or`. The following example shows how to create a number of queries:

CrudRepository (Example)

```
package hello;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

interface ProfileRepository extends CrudRepository<Profile, Long> {
    List<Profile> findByNameAndRole(String name, String role);

    // Enables the distinct flag for the query
    List<Profile> findDistinctNameByNameOrRole(String name, String role);
    List<Profile> findProfileDistinctByNameOrRole(String name, String role);

    // Enabling ignoring case for an individual property
    List<Profile> findByNameIgnoreCase(String name);
    // Enabling ignoring case for all suitable properties
    List<Profile> findByNameAndRoleAllIgnoreCase(String name, String role);

    // Enabling static ORDER BY for a query
    List<Profile> findByNameOrderByRoleAsc(String name);
    List<Profile> findByNameOrderByRoleDesc(String name);
}
```

Pagination

```
package hello;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.repository.CrudRepository;

import hello.User;

// This will be AUTO IMPLEMENTED by Spring into a Bean called userRepository
// CRUD refers Create, Read, Update, Delete

public interface UserRepository extends CrudRepository<User, Long> {

    Iterable <User> findByEmail(String email);
    Page <User> findByEmail(String email, Pageable pageable);
    Iterable <User> findByName(String name);
    Iterable <User> getByEmail(String email);
}
```

Pagination

page	Page you want to retrieve.
size	Size of the page you want to retrieve.
sort	Properties that should be sorted by in the format <code>property,property(,ASC DESC)</code> . Default sort direction is ascending. Use multiple <code>sort</code> parameters if you want to switch directions, e.g. <code>?sort=firstname&sort=lastname,asc</code> .

ตัวอย่าง url

`http://localhost:8080/demo/all?page=0&size=10&sort=email,asc`

วิธีการเรียกใช้ใน code

```
@GetMapping(path="/byEmail")
public @ResponseBody Page<User> getByEmail(
    @RequestParam String email,
     Pageable page
) {
    System.out.println(userRepository.findByName("First"));

    return userRepository.findByEmail(email, page);
}
```

Limiting Query Results

```
User findFirstByOrderByLastnameAsc();  
  
User findTopByOrderByAgeDesc();  
  
Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);  
  
Slice<User> findTop3ByLastname(String lastname, Pageable pageable);  
  
List<User> findFirst10ByLastname(String lastname, Sort sort);  
  
List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

Microservice with

SPRING BOOT

service admin with

SPRING BOOT ADMIN

Server Setup

pom.xml

...

```
<dependency>  
    <groupId>de.codecentric</groupId>  
    <artifactId>spring-boot-admin-starter-server</artifactId>  
    <version>2.1.5</version>  
</dependency>
```

...

Enable Admin Server

```
package com.business.adminserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import de.codecentric.boot.admin.server.config.EnableAdminServer;

@SpringBootApplication
@EnableAdminServer
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Client Setup

pom.xml

...

```
<dependency>  
    <groupId>de.codecentric</groupId>  
    <artifactId>spring-boot-admin-starter-client</artifactId>  
    <version>2.1.5</version>  
</dependency>
```

...

application.properties

```
spring.boot.admin.client.url=http://localhost:8080
```

Actuator

- Actuator is mainly used to expose operational information about the running application— health, metrics, info, dump, env, etc. It uses HTTP endpoints or JMX beans to enable us to interact with it.
- Once this dependency is on the classpath several endpoints are available for us out of the box. As with most Spring modules, we can easily configure or extend it in many ways

pom.xml

...

```
<dependency>  
    <groupId>de.codecentric</groupId>  
    <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>
```

...

application.properties

```
management.endpoints.web.exposure.include=*
```

centralize config with

CONFIG SERVER

pom.xml

...

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-config-server</artifactId>  
</dependency>
```

...

application.properties

server.port=8888

spring.cloud.config.server.git.uri=<GIT URI>

spring.cloud.config.server.git.username=<USER NAME>

spring.cloud.config.server.git.password=<PASSWORD>

Enable Config Server

```
package com.example.configserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

Git config file name

- file config ที่อยู่บน git ต้องอยู่ในรูปแบบ
 - <service_name>.properties
 - <service_name>.yml
- ต้องที่ client มา load config ไป จะดูที่ ชื่อ service name ที่ถูกตั้งค่าไว้ใน เช่น ใน project demo เราตั้งชื่อ service ว่า
 - spring.application.name=demo-service
- ต้องที่ start service demo-service มันจะอ่านมาค้นหา file **demo-service.properties** ใน config server

use config from server with

CONFIG CLIENT

pom.xml

...

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-starter-config</artifactId>  
</dependency>
```

...

bootstrap.properties

spring.application.name=demo-service

spring.cloud.config.uri=http://localhost:8888

Microservice with

EUREKA SERVER

pom.xml

...

```
<dependency>  
    <groupId>org.springframework.cloud</groupId>  
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>  
</dependency>
```

...

application.properties

```
# Give a name to the eureka server
spring.application.name=eureka-server

# default port for eureka server
server.port=8761

# eureka by default will register itself as a client. So, we need to set it to false.
# What's a client server? See other microservices (image, gallery, auth, etc).
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

Enable EurekaServer

```
package com.eureka.server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer // Enable eureka server

public class SpringEurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringEurekaServerApplication.class, args);
    }
}
```

Service Discovery with

EUREKA DISCOVERY

pom.xml

...

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>  
</dependency>
```

...

application.properties

spring.application.name=demo-service

server.port=8200

eureka.client.service-url.default-zone=http://localhost:8761/eureka

##FOR FIXED IP AND HOST NAME

spring.cloud.client.hostname=localhost

spring.cloud.client.ipAddress=127.0.0.1

eureka.instance.hostName=localhost

eureka.instance.preferIpAddress=false

Enable EurekaClient

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}

@Bean
@LoadBalanced
public RestTemplate restTemplate(RestTemplateBuilder builder) {
    return builder.build();
}
```

Call to another service

@Autowired

```
private RestTemplate restTemplate;
```

```
@RequestMapping("/persons")
```

```
public void getPersons() {
```

```
    List<Object> persons = restTemplate.getForObject("http://demo/  
persons/", List.class);
```

```
}
```

Cloud Routing with

ZUUL

pom.xml

....

```
<dependency>  
  <groupId>org.springframework.cloud</groupId>  
  <artifactId>spring-cloud-starter-netflix-zuul</artifactId>  
</dependency>
```

....

application.properties

```
server.port=8762
spring.application.name=zuul-server
eureka.client.service-url.default-zone=http://localhost:8761/eureka/
```

```
# A prefix that can be added to beginning of all requests.
```

```
#zuul.prefix=/api
```

```
# Disable accessing services using service name (i.e. gallery-service).
```

```
# They should be only accessed through the path defined below.
```

```
zuul.ignored-services=*
```

```
# Map paths to services
```

```
zuul.routes.XXX.path=/x/**
```

```
zuul.routes.XXX.service-id=demo-service
```

```
zuul.routes.b.path=/bbb/**
```

Enable Zuul Proxy

```
package com.comp.zuul.zuulserver;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.netflix.zuul.EnableZuulProxy;

@SpringBootApplication
@EnableZuulProxy
@EnableEurekaClient
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

security with

SPRING SECURITY

JWT

- JSON Web Token (JWT) is an open standard ([RFC 7519](#)) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA** or **ECDSA**.
- Although JWTs can be encrypted to also provide secrecy between parties, we will focus on `signedtokens`. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

setup proxy server [zuul]

pom.xml

....

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

....

application.properties

Map path to auth service

zuul.routes.auth-service.path=/auth/**

zuul.routes.auth-service.service-id=AUTH-SERVICE

strip-prefix to false

zuul.routes.auth-service.strip-prefix=false

Exclude authorization from sensitive headers

zuul.routes.auth-service.sensitive-headers=Cookie,Set-Cookie

SecurityTokenConfig

```
package com.comp.zuul.zuulserver;

import javax.servlet.http.HttpServletResponse;
import org.springframework.http.HttpMethod;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@EnableWebSecurity
public class SecurityTokenConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .csrf().disable()
            // make sure we use stateless session; session won't be used to store user's state.
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            // handle an authorized attempts
            .exceptionHandling().authenticationEntryPoint((req, rsp, e) -> rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED))
            .and()
            // Add a filter to validate the tokens with every request
            .addFilterAfter(new JwtTokenAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class)
        // authorization requests config
        .authorizeRequests()
            // allow all who are accessing "auth" service
            .antMatchers(HttpMethod.POST, "").permitAll()
            // must be an admin if trying to access admin area (authentication is also required here)
            .antMatchers("/demo" + "/admin/**").hasRole("ADMIN")
            // Any other request must be authenticated
            .anyRequest().authenticated();
    }
}
```

JwtTokenAuthenticationFilter

```
package com.comp.zuul.zuulserver;

import java.io.IOException;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.filter.OncePerRequestFilter;

public class JwtTokenAuthenticationFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException {
        /**
         * YOUR BUSINESS LOGIC HERE
         */
    }
}
```

setup auth service

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.0</version>
</dependency>
```

application.properties

spring.application.name=auth-service

server.port=9100

eureka.client.service-url.default-zone=http://localhost:8761/eureka

Implement Class

- WebSecurityConfigurerAdapter
- UserDetailsService
- UsernamePasswordAuthenticationFilter

ldap

```
<dependency>
    <groupId>org.springframework.ldap</groupId>
    <artifactId>spring-ldap-core</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-ldap</artifactId>
</dependency>

<dependency>
    <groupId>com.unboundid</groupId>
    <artifactId>unboundid-ldapsdk</artifactId>
</dependency>
```

Example code (java)

```
@Override
public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .ldapAuthentication()
        .userDnPatterns("uid={0},ou=people")
        .groupSearchBase("ou=groups")
        .contextSource()
        .url("ldap://localhost:8389/dc=springframework,dc=org")
        .and()
        .passwordCompare()
        .passwordEncoder(new LdapShaPasswordEncoder())
        .passwordAttribute("userPassword");
}
```

Circuit Breaker with

HYSTRIX

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

@HystrixCommand annotation

```
@HystrixCommand(fallbackMethod = "fallback")
@RequestMapping("/{id}")
public void getPersons() {
    List<Object> persons = restTemplate.getForObject("http://demo/
persons/", List.class);
}

public void fallback(Throwable hystrixCommand) {
    System.out.println("get Person error")
}
```

Log Tracing with
SLEUTH

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
```

Logger

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

// ...
private static final Logger LOGGER =
LoggerFactory.getLogger(HomeController.class);

public Gallery getGallery(@PathVariable final int id) {
    LOGGER.info("Creating gallery object ... ");
    // ...
    LOGGER.info("Returning images ... ");
    return images;
}
// ...
```

trace and span ids

ชื่อ application Tracing ID Span ID Flag กับบอกว่าส่ง data
ไป zipkin หรือไม่

```
2018-08-05 14:08:30.623 INFO [pricing-service,76bf8bf9adf1312f,76bf8bf9adf1312f,false] 10573 ---  
[nio-9000-exec-9] c.e.PricingService.PricingController : get Price by SKU : bnk48  
2018-08-05 14:08:31.175 INFO [pricing-service,df9c6d3ec6a63e14,df9c6d3ec6a63e14,false] 10573 ---  
[io-9000-exec-10] c.e.PricingService.PricingController : get Price by SKU : bnk48
```

distributed tracing system with

ZIPKIN

Server Setup

docker-compose.yml

```
version: '3.6'

services:
  zipkin:
    image: openzipkin/zipkin
    ports:
      - 9411:9411
    container_name: zipkin
```

start server

- run docker-compose command

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

Client Setup

pom.xml

...

```
<dependency>  
    <groupId>de.codecentric</groupId>  
    <artifactId>spring-cloud-starter-zipkin</artifactId>  
</dependency>
```

...

application.properties

spring.zipkin.baseUrl=http://localhost:9411

spring.sleuth.sampler.probability=1.0

หมายเหตุ spring.sleuth.sampler.percentage property ชี้งจะรับค่า 0 กึ่ง 1 โดยหมายถึงจำนวน % ที่จะส่ง log ไปที่ zipkin โดย 1 = 100%

centralized logging with

ELK

Dockerfile

```
FROM sebp/elk

ADD ./02-beats-input.conf /etc/logstash/conf.d/02-beats-input.conf
ADD ./30-output.conf /etc/logstash/conf.d/30-output.conf
```

30-output.conf

```
filter {  
    json {  
        source => "message"  
    }  
}  
  
output {  
    elasticsearch {  
        hosts => ["localhost"]  
        manage_template => false  
        index => "elk-%{+YYYY.MM.dd}"  
    }  
}
```

02-beats-input.conf

```
input {  
    tcp {  
        port => 5044  
    }  
}
```

Build and run

- build command

```
docker build --tag elk .
```

- run command

```
docker run -p 5601:5601 -p 9200:9200 -p 5044:5044 --name elk elk
```

== server url==

Kibana => <http://localhost:5601/app/kibana>

Elastic Search => <http://localhost:9200>

Logstash => localhost:5044

Configure Logback

pom.xml

...

```
<dependency>
    <groupId>net.logstash.logback</groupId>
    <artifactId>logstash-logback-encoder</artifactId>
    <version>5.1</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-classic</artifactId>
    <version>1.2.3</version>
</dependency>
<dependency>
    <groupId>ch.qos.logback</groupId>
    <artifactId>logback-core</artifactId>
    <version>1.2.3</version>
</dependency>
...
```

logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration debug="false">
    <include resource="org/springframework/boot/logging/logback/base.xml"/>
    <appender name="logstash" class="net.logstash.logback.appenders.LogstashTcpSocketAppender">
        <destination>localhost:5044</destination>
        <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
            <providers>
                <pattern>
                    <pattern>{"appName": "elk-pricing-service"}</pattern>
                </pattern>
                <threadName/>
                <stackTrace/>
            </providers>
        </encoder>
    </appender>
    <root level="INFO">
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="logstash"/>
    </root>
    <logger name="org.springframework" level="INFO"/>
</configuration>
```

Any questions?

