Spring Boot: Efficient Development, Configuration, and Deployment

LEVERAGING INITIALIZR AND DEVTOOLS FOR EFFICIENT DEVELOPMENT

Course Overview

You know the basics of Spring Boot, now how can you be better at it?

Faster

Smarter

Easier

“Cloudier”

Faster

- Generate project scaffolding using Initializr and your IDE
- Spring Boot startup from 10 seconds to 2 during development

Smarter

- Learn how @EnableAutoConfiguration works
  - Tune or disable to your needs
  - Write your own

Easier

- Streamline the configuration of your application
  - Relaxed bindings
  - YAML
  - Eliminate casting using typesafe @ConfigurationProperties
  - Integrate with your IDE

”Cloudier”

- Deploy your Spring Boot apps to the cloud using Docker
- Monitor your app in the cloud or on your servers using built in Actuators
- Write your own custom monitoring Actuators

cURL - Download

https://curl.haxx.se/download.html

Win32 - cygwin

cygwin	7.47.0	**binary**	SSL		Cygwin	272 KB
cygwin	7.47.0	libcurl	SSL		Cygwin	184 KB

Win64 - Generic

Win64 ia64 zip	7.33.0	**binary**			Don Luchini	228 KB
Win64 ia64 zip	7.33.0	**binary**	SSL		Don Luchini	697 KB
Win64 ia64 CAB	7.47.1	**binary**	SSL		Stefan Kanthak	
Win64 x86_64 CAB	7.47.1	**binary**	SSL		Stefan Kanthak	
Win64 x86_64 7zip	7.47.1	**binary**	SSL		Darren Owen	
Win64 x86_64 7zip	7.47.1	**binary**	SSL	SSH	Viktor Szakáts	1.82 MB
Win64 2000/XP x86_64 MSI	7.46.0	**binary**	SSL	SSH	Edward LoPinto	
Win64 2000/XP x86_64 zip	7.46.0	**binary**	SSL	SSH	Edward LoPinto	

Win64 - cygwin

cygwin	7.47.0	**binary**	SSL		Cygwin	272 KB
cygwin	7.47.0	libcurl	SSL		Cygwin	175 KB

Win64 - MinGW64

MinGW64	7.40.0	**binary**	SSL	SSH	Günter Knauf	1.19 MB
MinGW64	7.40.0	devel	SSL	SSH	Günter Knauf	2.21 MB

z/OS

z/OS	7.16.1	**binary**	SSL		IBM

This colour means the packaged version is the latest stable version available (7.47.1)!

More information on **metalink** downloads is available from www.metalinker.org.

If you have newer archives or archives for platforms not already present in this table, we'd like to add them to this table with a pointer to your location. Mail curl-release and tell us!

Downloads

File Home Share View

Pin to Quick access | Copy | Paste | Cut | Copy path | Paste shortcut | Move to | Copy to | Delete | Rename | New folder | New item | Easy access | Properties | Open | Edit | History | Select all | Select none | Invert selection

Clipboard | Organize | New | Open | Select

This PC › Downloads

Search Downloads

- Quick access
- Desktop
- Downloads
- Documents
- Pictures
- Music
- skins
- Videos
- OneDrive
- This PC
- Network
- Homegroup

Name	Date modified	Type	Size
curl-7.47.1	2/21/2016 9:02 PM	Cabinet File	1,420 KB

1 item

Command Prompt

```
c:\dev>curl start.spring.io

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

:: Spring Initializr ::  https://start.spring.io

This service generates quickstart projects that can be easily customized.
Possible customizations include a project's dependencies, Java version, and
build system or build structure. See below for further details.

The services uses a HAL based hypermedia format to expose a set of resources
to interact with. If you access this root resource requesting application/json
as media type the response will contain the following links:
+----------------+----------------------------------------+
| Rel            | Description                            |
+----------------+----------------------------------------+
| gradle-build   | Generate a Gradle build file           |
| gradle-project | Generate a Gradle based project archive |
| maven-build    | Generate a Maven pom.xml               |
| maven-project * | Generate a Maven based project archive |
+----------------+----------------------------------------+


The URI templates take a set of parameters to customize the result of a request
to the linked resource.
+----------------+----------------------------------------+-----------------------------+
| Parameter      | Description                            | Default value               |
+----------------+----------------------------------------+-----------------------------+
| applicationName | application name                       | DemoApplication             |
| artifactId     | project coordinates (infer archive name) | demo                        |
| baseDir        | base directory to create in the archive | no base dir                 |
| bootVersion    | spring boot version                    | 1.3.2.RELEASE               |
| dependencies   | dependency identifiers (comma-separated) | none                        |
| description    | project description                    | Demo project for Spring Boot |
| groupId        | project coordinates                    | com.example                 |
| javaVersion    | language level                         | 1.8                         |
| language       | programming language                   | java                        |
| name           | project name (infer application name)  | demo                        |
```

Finishing Your Just Gif It Application

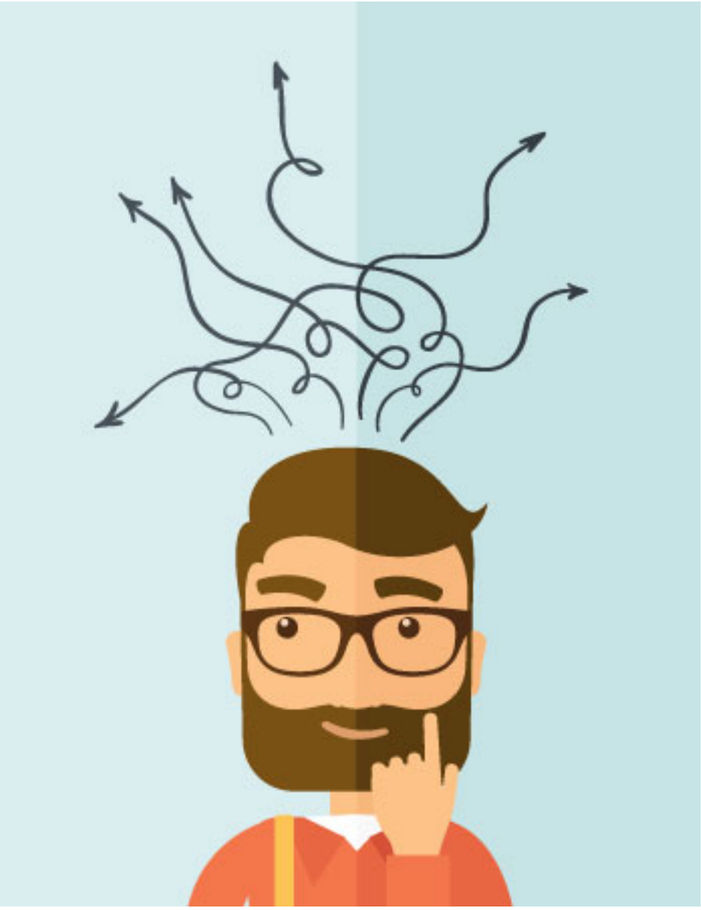
1. Extract on top of existing code
2. Run: mvn clean install

- Intro to the demo application
- Spring Initialzr
- Spring Boot support in the IDE
- Spring Boot Devtools

Reducing Code with Custom Auto Configurations

Smarter

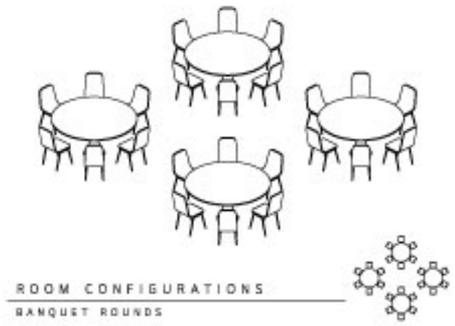
- @EnableAutoConfiguration
  - Review
  - Tuning
  - Demystifying
  - Writing our own



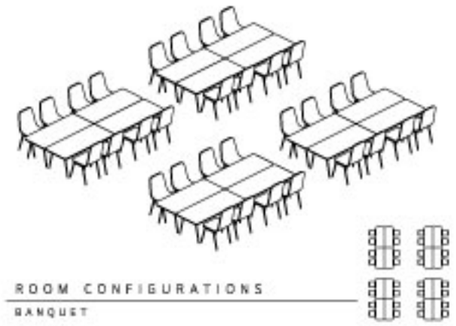
Spring Framework is highly configurable

- Web?
- Front end?
- Data access?
- Security?

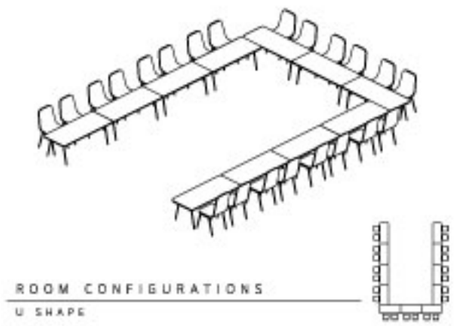
Decisions. So many decisions.



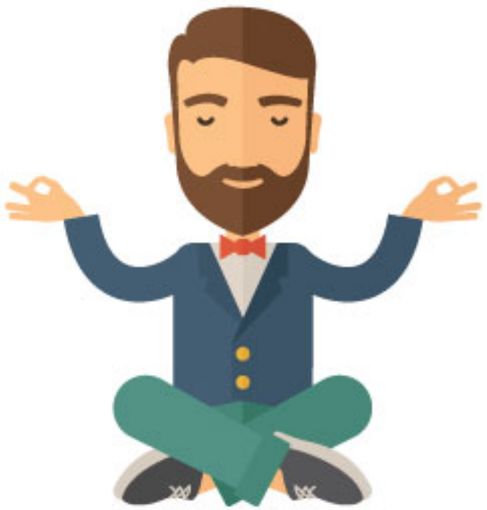
Banquet Style Round



Banquet Style Square



U Shape Configuration



`@EnableAutoConfiguration` **is like having your own opinionated event planner.**

Phil Webb
@phillip_webb

Follow

For those on reddit.com/r/java/ saying it's Spring Boot is "the framework for a framework" here's a diagram:

RETWEETS 112 LIKES 66

7:54 PM - 8 Sep 2015

Spring Boot is **not** Spring Boot without Auto Configurations!

```java
package schultz.dustin.io

@EnableAutoConfiguration
public class Foo {
    ...
}
```

- Package has significance
- Intelligent and seemingly “magical” annotation that enables features and configures functionality

```
package schultz.dustin.io

@Configuration
@ComponentScan
@EnableAutoConfiguration
public class Foo {
    ...
}
```

3 very common annotations in Spring Boot apps

```java
package schultz.dustin.io

@SpringBootApplication
public class Foo {
    ...
}
```

- Introduced in Spring Boot 1.2. Really three annotations in one.

Tuning your Auto Configuration

Intelligent Decision Making Based on Conditions

Presence/
Absence of Jar

Presence/
Absence of Bean

Presence/Absence
of Property

Many More!

==========

AUTO-CONFIGURATION REPORT

==========

```
Positive matches:
-----------------

EmbeddedServletContainerAutoConfiguration
matched
      - found web application
StandardServletEnvironment
(OnWebApplicationCondition)

...
```

```
Negative matches:
-----------

CassandraAutoConfiguration did not match
      - required @ConditionalOnClass classes
not found: com.datastax.driver.core.Cluster
(OnClassCondition)

...
```

```
Exclusions:
-----------

org.springframework.boot.autoconfigure.
jdbc.DataSourceAutoConfiguration
```

Unconditional classes:

---

org.springframework.boot.autoconfigure.
PropertyPlaceholderAutoConfiguration

...

Enabling the Auto Configuration Report

`--debug` cmd line arg	`-Ddebug` VM arg	`export DEBUG=true` Environment var
`debug=true` application.properties	`logging.level.=debug` application.properties	**Many more!**

You’ve identified an auto configuration that needs tuning, now what?

```
1 @EnableAutoConfiguration(exclude = { SomeAutoConfig.class })
2 public class JustGifItApplication {
3     ...
4 }
```

Excluding Unnecessary or Misconfigured Auto Configurations via Annotations

Via the exclude **parameter of annotation as a comma separated list of classes**

- http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#auto-configuration-classes

```
1 @EnableAutoConfiguration(excludeName = { “a.b.SomeAutoConfig” })
2 public class JustGifItApplication {
3     ...
4 }
```

Excluding Unnecessary or Misconfigured Auto Configurations via Annotations

Via the excludeName **parameter of annotation as a comma separated list of class names**

- http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#auto-configuration-classes

```
1 @SpringBootApplication(exclude = { SomeAutoConfig.class })
2 public class JustGifItApplication {
3     ...
4 }
```

Excluding Unnecessary or Misconfigured Auto Configurations via Annotations

Via the exclude **parameter of annotation as a comma separated list of classes**

- http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#auto-configuration-classes

```
1 @SpringBootApplication(excludeName = { “a.b.SomeAutoConfig” })
2 public class JustGifItApplication {
3     ...
4 }
```

Excluding Unnecessary or Misconfigured Auto Configurations via Annotations

Via the excludeName **parameter of annotation as a comma separated list of classes**

- http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#auto-configuration-classes

application.properties

```
...
spring.autoconfigure.exclude = my.company.SomeAutoConfig
...
```

Excluding Unnecessary or Misconfigured Auto Configurations via Properties

Via the application.properties **as a comma separated list of classes**

* Can be used in addition to annotations

When to Exclude Auto Configurations

- **Exclusions are all or nothing**
- **Use when you don’t need configuration at all**

application.properties

```
# DAO (PersistenceExceptionTranslationAutoConfiguration)
spring.dao.exceptiontranslation.enabled = false

# Spring MVC
spring.mvc.favicon.enabled = false
```

Reconfiguring Auto Configurations via Properties (`@ConditionalOnProperty`)

Tune auto configurations via properties in the application.properties

* http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#common-application-properties

```
PropertySourcesPlaceholderConfigurer#propertySourcesPlaceHolderConfigurer
   @ConditionalOnMissingBean (types: o.s.c.s.PropertySourcesPlaceholderConfigurer)
```

```java
@Bean
// Define our own
public static PropertySourcesPlaceholderConfigurer
propertySourcesPlaceholderConfigurer() {
    PropertySourcesPlaceholderConfigurer placeholderConfigurer
     = new PropertySourcesPlaceholderConfigurer();
    placeholderConfigurer.setNullValue("");
    return placeholderConfigurer;
}
```

Overriding @ConditionalOnMissingBean Conditions

By defining a new @Bean **before Auto Configuration happens**

```
1 @Bean
2 // Disable HiddenHttpMethodFilter
3 public FilterRegistrationBean
4  register(HiddenHttpMethodFilter hiddenHttpMethodFilter) {
5      FilterRegistrationBean bean
6       = new FilterRegistrationBean(hiddenHttpMethodFilter);
7      bean.setEnabled(false);
8      return bean;
9 }
```

Excluding Auto Configured Servlets or Filters

By defining a `FilterRegistrationBean` **or a** `ServletRegistrationBean` **and calling** `setEnabled(false)`

```
1 @Configuration
2 @Import({
3         DispatcherServletAutoConfiguration.class,
4         EmbeddedServletContainerAutoConfiguration.class,
5         ...
6 })
7 public class JustGifItApplication {
8         ...
9 }
```

Completely Custom Auto Configuration

Remove `@EnableAutoConfiguration` **and manually configure an array of auto configuration classes with the** `@Configuration` **and** `@Import` **annotations**

When in doubt, always check the Spring Documentation first

Using the @ConditionalOn... Annotations

Spring 4 Conditional Configuration

A single condition that must be matched in order for a component to be registered.

Presence or Absence of Class on Classpath

@ConditionalOnClass

	Attributes	
✓	[]	name = { "a.b.c.Foo" }
✓	[]	value = { Foo.class }

@ConditionalOnMissingClass

	Attributes	
✓	[]	name = { "a.b.c.Foo" }
✗	[]	~~value = { Foo.class }~~

Presence or Absence of Defined Bean

@ConditionalOnBean

	Attributes	
✓	[ ]	name = { “dataSource” }
✓	[ ]	value = { Foo.class }
✓	[ ]	type = { “a.b.c.Foo” }
✓	[ ]	annotation = { Foo.class }
✓	enum	search = { ALL }
✗	[ ]	~~ignored = { “a.b.c.Foo” }~~
✗	[ ]	~~ignoredType = { Foo.class }~~

@ConditionalOnMissingBean

	Attributes	
✓	[ ]	name = { “dataSource” }
✓	[ ]	value = { Foo.class }
✓	[ ]	type = { “a.b.c.Foo” }
✓	[ ]	annotation = { Foo.class }
✓	enum	search = { ALL }
✓	[ ]	ignored = { “a.b.c.Foo” }
✓	[ ]	ignoredType = { Foo.class }

Presence or Absence of a Property Having Value

@ConditionalOnProperty

	Attributes	
	[]	name = { “my-property” }
	[]	value = { “my-property” }
	String	havingValue = “foo”
	[]	prefix = { “some.prefix” }
	boolean	matchIfMissing = false
	boolean	relaxedNames = true

Additional Conditions

- @ConditionalOnJava
- @ConditionalOnJndi
- @ConditionalOnResource
- @ConditionlOnExpression
- @ConditionlOnWebApplication
- @ConditionalOnNotWebApplication
- Don’t be afraid to mix and match!

Writing Our First Auto Configuration

http://github.com/dustinschultz/just-gif-it-autoconfigure

Demystifying Auto Configuration

@EnableAutoConfiguration Unveiled

```
@EnableAutoConfiguration
```

```
@Import(EnableAutoConfigurationImportSelector.class)
```

```
SpringFactoriesLoader.loadFactoryNames(...)
```

```
/META-INF/spring.factories
```

```
o.s.b.a.EnableAutoConfiguration = o.s.b.a.SomeAutoConfig, ...
```

Module in Review

- **Auto configuration report**
- **Tuning auto configuration**
- `@ConditionalOn`**... annotations**
- **Writing our own custom auto configurations**

Making Use of Spring Boot's Improvements to Externalized Configuration

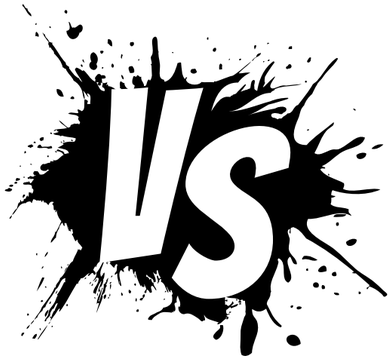
Easier

- Enhanced configuration
  - YAML
  - Typesafe configuration
  - Resolving configuration

YAML™

A data serialization standard made for configuration files!

- Pronounced YAM-EL, rhymes with Camel
- Data serialization language / format
- Since 2001
- Ruby, Python, ElasticSearch, MongoDb

YAML  .properties

- **Defined spec: http://yaml.org/spec/**
- Human readable
- **key/value (Map), Lists, and Scalar types**
- Used in many languages
- **Hierarchical**
- Doesn't work with `@PropertySource`
- **Multiple Spring Profiles in default config**

- **java.util.Properties Javadoc is spec**
- Human readable
- **key/value (Map) and String types**
- Used primarily in Java
- **Non-hierarchical**
- Works with `@PropertySource`
- **One Spring Profile per config**

.properties

```
some_key=value
some_number=9
some_bool=true
```

.yml

```
some_key: value
some_number: 9

# could use values yes or on
some_bool: true
```

🔑 **YAML Basics**: Key/Value Scalars

- `.properties` **keys and values are Strings**
- `.yml` **keys are Strings and values are their respective type**

.properties

```
# A map
somemap.key=value
somemap.number=9

# Another map
map2.bool=true
map2.date=2016-01-01
```

.yml

```
# A map
somemap:
  key: value
  number: 9

# Inline map
map2: {bool=true,date=2016-01-01}
```

YAML Basics: Maps

- `.properties` **uses dots to denote hierarchy (a Spring convention)**
- `.yml` **uses hierarchy (consistent spaces) to create maps**

.properties

```
# A list
numbers[0]=one
numbers[1]=two

# Inline list
numbers=one,two
```

.yml

```
# A list
numbers:
  - one
  - two

# Inline list
numbers: [one,two]
```

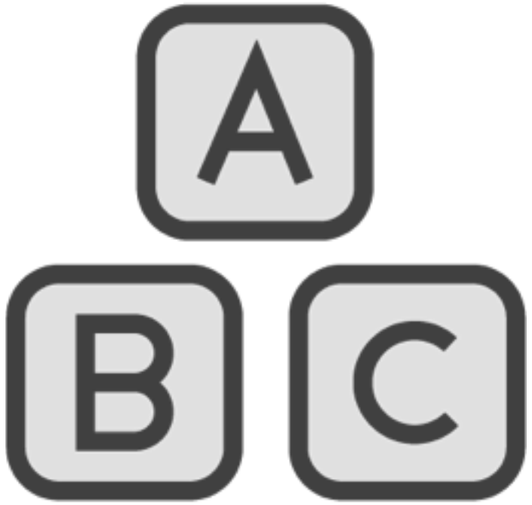
YAML Basics: Lists

- `.properties` **uses prop[index] or commas for a List (a Spring convention)**
- `.yml` **uses '`- value`' or commas surrounded with brackets for a List**

What should I use, Properties or YAML?

Typesafe Configuration

Getting Started with `@ConfigurationProperties`



a. **Annotate with** `@ConfigurationProperties`

b. **Define getters & setters (JavaBean Spec)**

c. **Annotate with** `@Component`

   i. **Can also use** `@EnableConfigurationProperties`

`@ConfigurationProperties` turns all of your application configuration into typesafe POJOs

.properties

```
my.feature-enabled=true
```

.yml

```yml
my:
  feature-enabled: true
```

Using the Above Configuration...

Annotate class with **@ConfigurationProperties**

```
1
2 @ConfigurationProperties(prefix = "my")
3 public class MyConfig
4 {
5
6
7
8
9
10
11
12
13
14 }
```

Create an Instance Variable for Your Property

```
1
2 @ConfigurationProperties(prefix = "my")
3 public class MyConfig
4 {
5     private Boolean featureEnabled;
6
7
8
9
10
11
12
13
14 }
```

Define a Getter and a Setter for Your Property

```
1
2 @ConfigurationProperties(prefix = "my")
3 public class MyConfig
4 {
5     private Boolean featureEnabled;
6
7     public Boolean getFeatureEnabled() {
8         return featureEnabled;
9     }
10
11    public void setFeatureEnabled(Boolean featureEnabled) {
12        this.featureEnabled = featureEnabled;
13    }
14 }
```

Annotate class with **@Component**

```
1  @Component
2  @ConfigurationProperties(prefix = "my")
3  public class MyConfig
4  {
5      private Boolean featureEnabled;
6
7      public Boolean getFeatureEnabled() {
8          return featureEnabled;
9      }
10
11     public void setFeatureEnabled(Boolean featureEnabled) {
12         this.featureEnabled = featureEnabled;
13     }
14 }
```

Or … Use **@EnableConfigurationProperties**

```
1 @SpringBootApplication
2 @EnableConfigurationProperties(MyConfig.class)
3 public class MyApplication {
4     ...
5 }
```

Autowire It into Any Class

```
1 @Service
2 public class MyService
3 {
4     @Inject
5     private MyConfig config;
6
7     ...
8 }
```

“Maps and Collections can be expanded with only a getter, whereas arrays require a setter.”

Spring Boot Reference Documentation

Configuring Your `@ConfigurationProperties`

	Attributes	
	boolean	exceptionIfInvalid = true
	boolean	ignoreInvalidFields = false
	boolean	ignoreNestedProperties = false
	boolean	ignoreUnknownFields = true
	[]	locations = ["..."]
	boolean	merge = true
	String	prefix|value="some.namespace"

Validating Your Configuration

- **Simply annotate your instance variables with JSR-303 Annotations**
  - `@NotNull`
  - `@Pattern`
  - `@Max`
  - `@Min`
  - `@Digits`
- **And more**

`@ConfigurationProperties` aren’t only limited to beans you create. You can use them to configure third party beans too!

```java
@Configuration
public class MyConfig
{
    @Bean
    @ConfigurationProperties(
      prefix = "config.some-bean")
    public SomeBean someBean()
    {
      // Has getters & setters
      return new SomeBean()
    }
}
```

application.properties

```properties
# someBean has setFirstName method
config.some-bean.first-name=Dustin

# someBean has setLastName method
config.some-bean.last-name=Schultz
```

Configuring Third Party Beans

Resolving Configuration

Relaxed Configuration Names

Camel Case

`featureEnabled`

Dash Notation

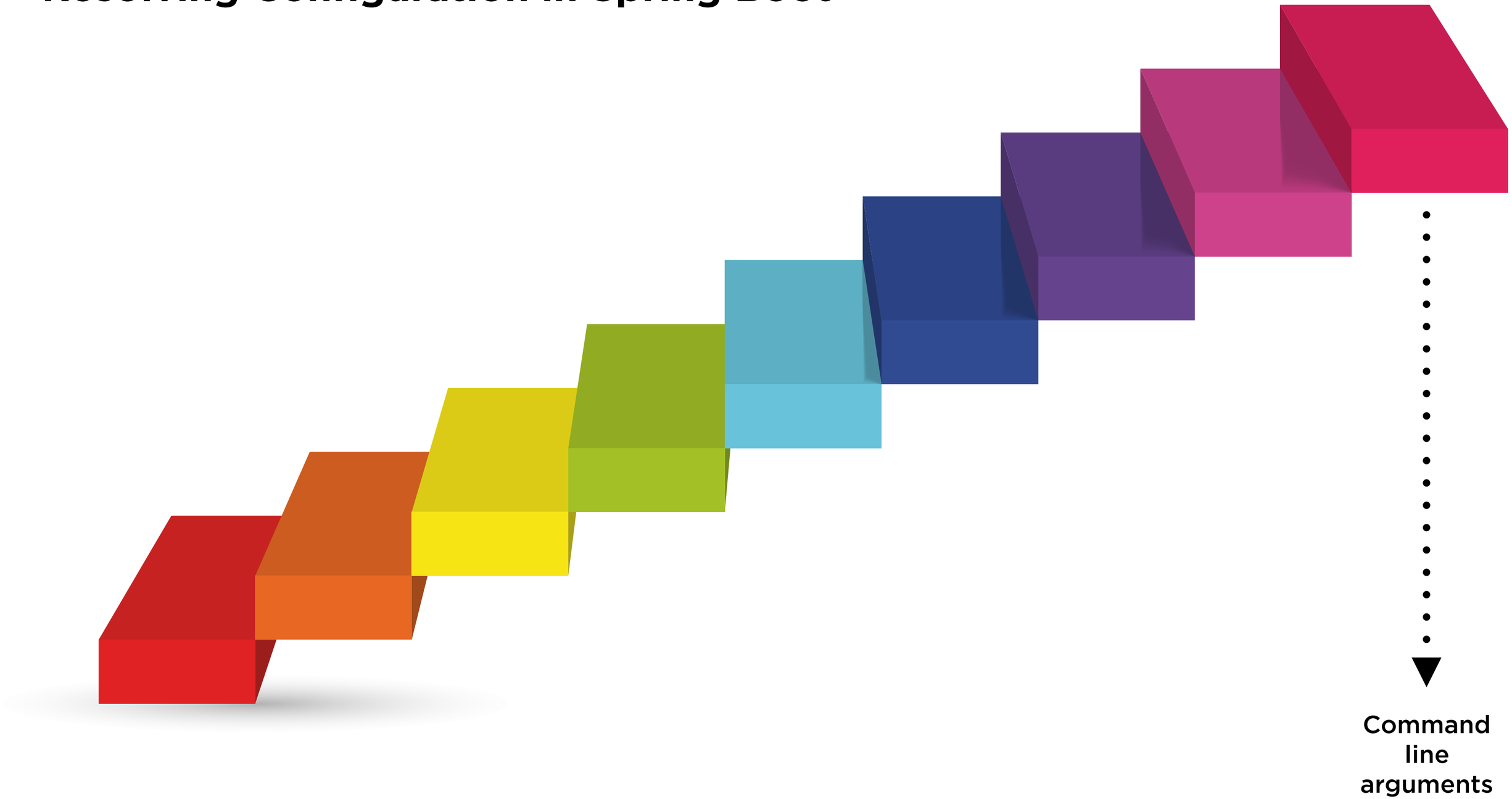
`feature-enabled`

Underscore

`PREFIX_FEATURE_ENABLED`

Spring Boot provides a standard cascading resolution of configuration.

Resolving Configuration in Spring Boot

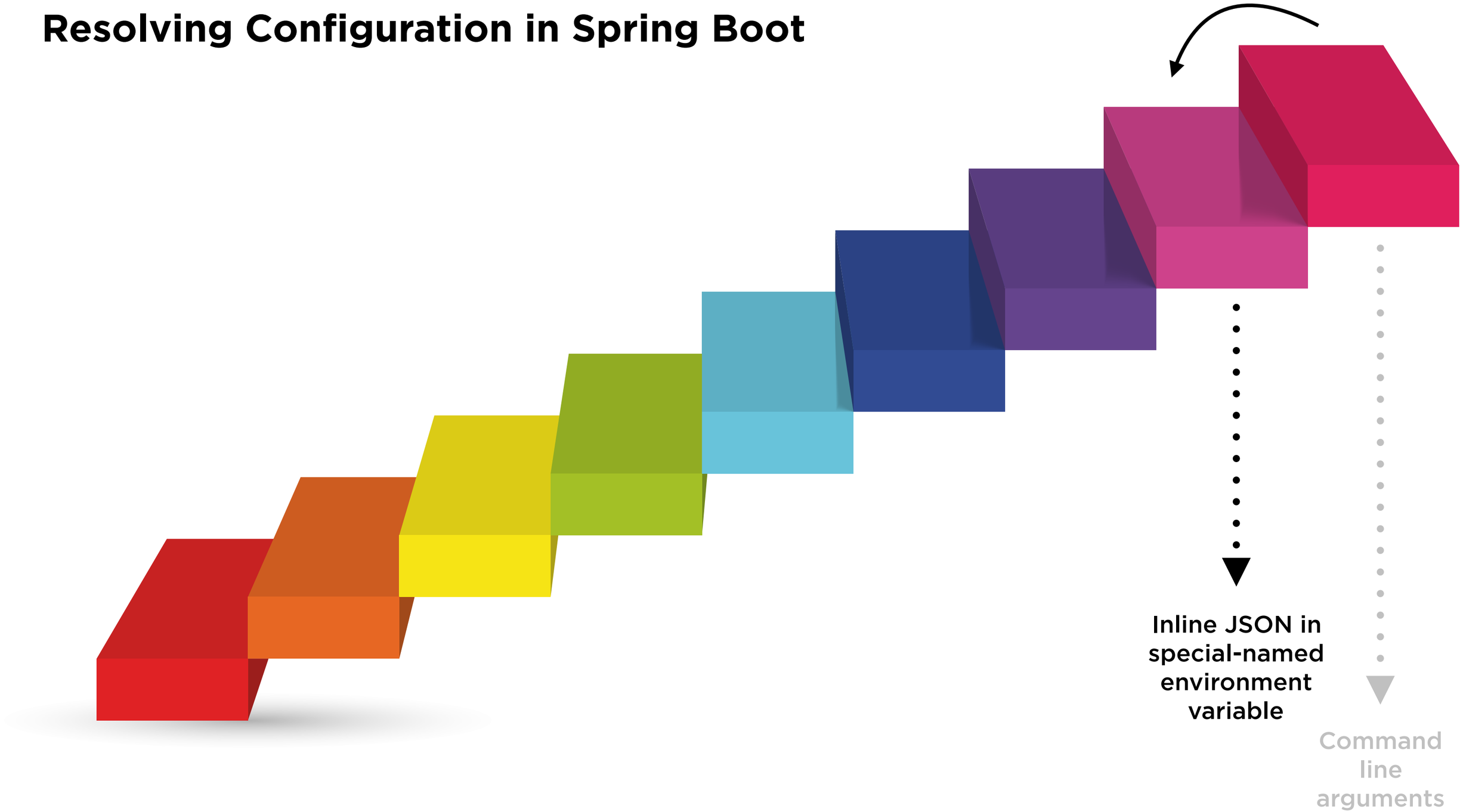


1.) Command Line Arguments

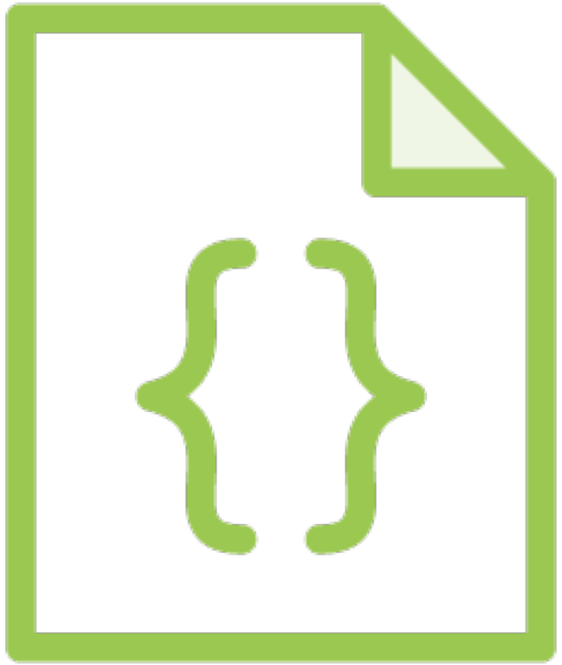
- **Prefix any property with a double dash**

```
--server.port=9000
--spring.config.name=config
--debug
```

Resolving Configuration in Spring Boot

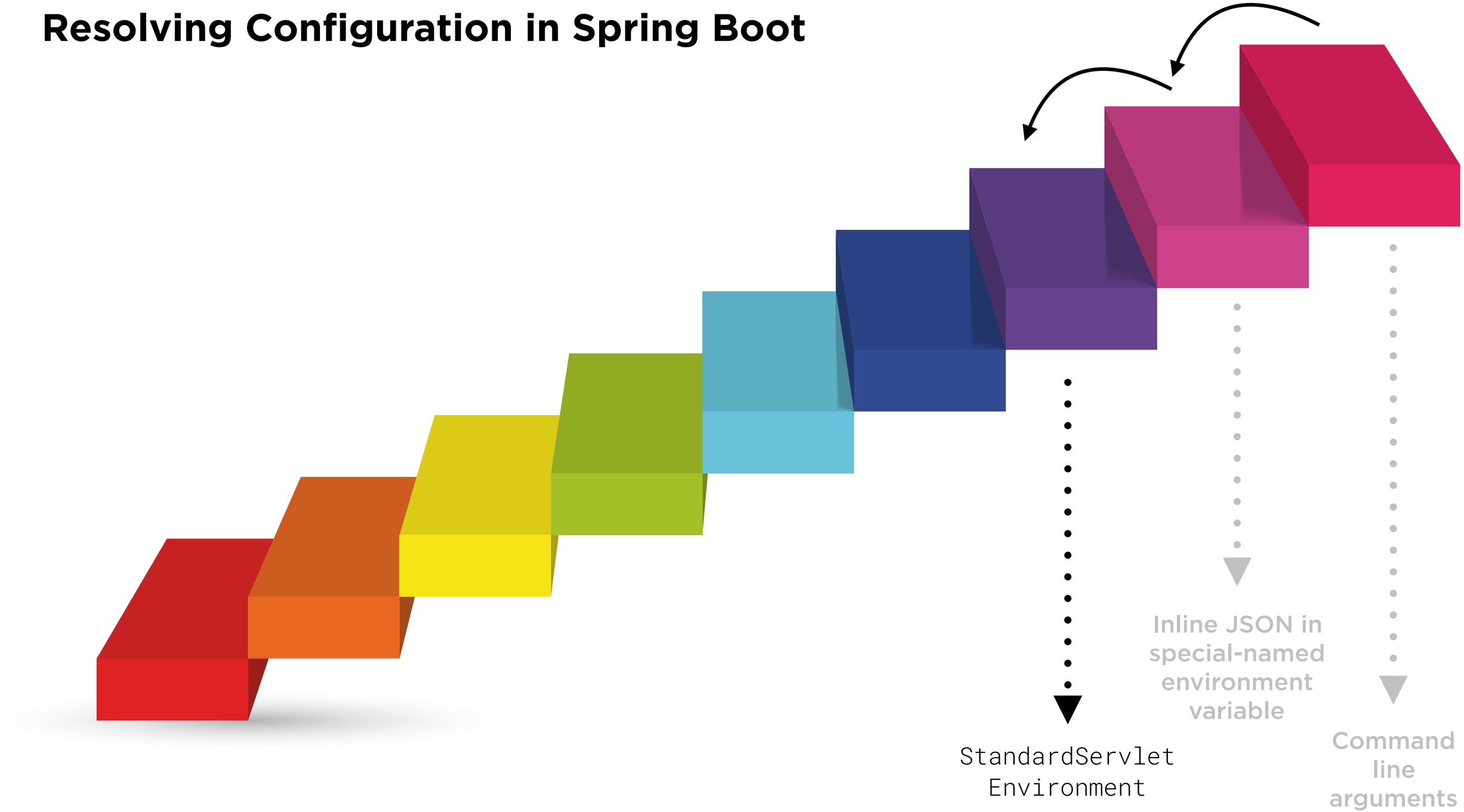


2.) Embedded JSON in SPRING_APPLICATION_JSON



- **SPRING_APPLICATION_JSON**=<JSON_STRING>
  - e.g. **SPRING_APPLICATION_JSON**=
    '{"server":{"port":"9000"}}'

Resolving Configuration in Spring Boot



3.) `StandardServletEnvironment`

- **A hierarchy within itself**

  a) `ServletConfig` **init parameters**

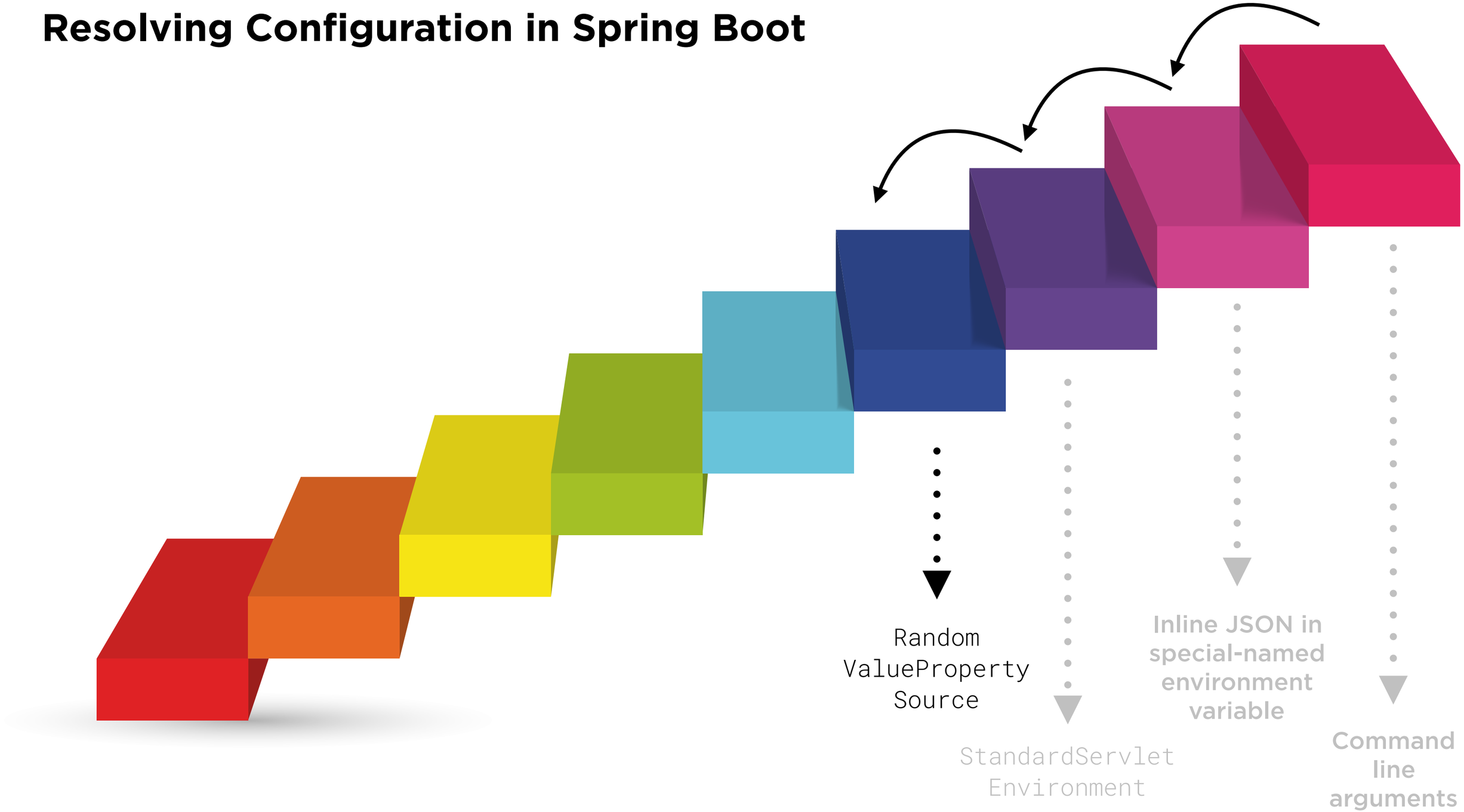
  b) `ServletContext` **init parameters**

  c) JNDI attributes

  d) System.getProperties()

  e) OS environment vars

Resolving Configuration in Spring Boot



4.) RandomValuePropertySource

- **${random.*} replacements**
- **“ * ” can be one of**

  A. value

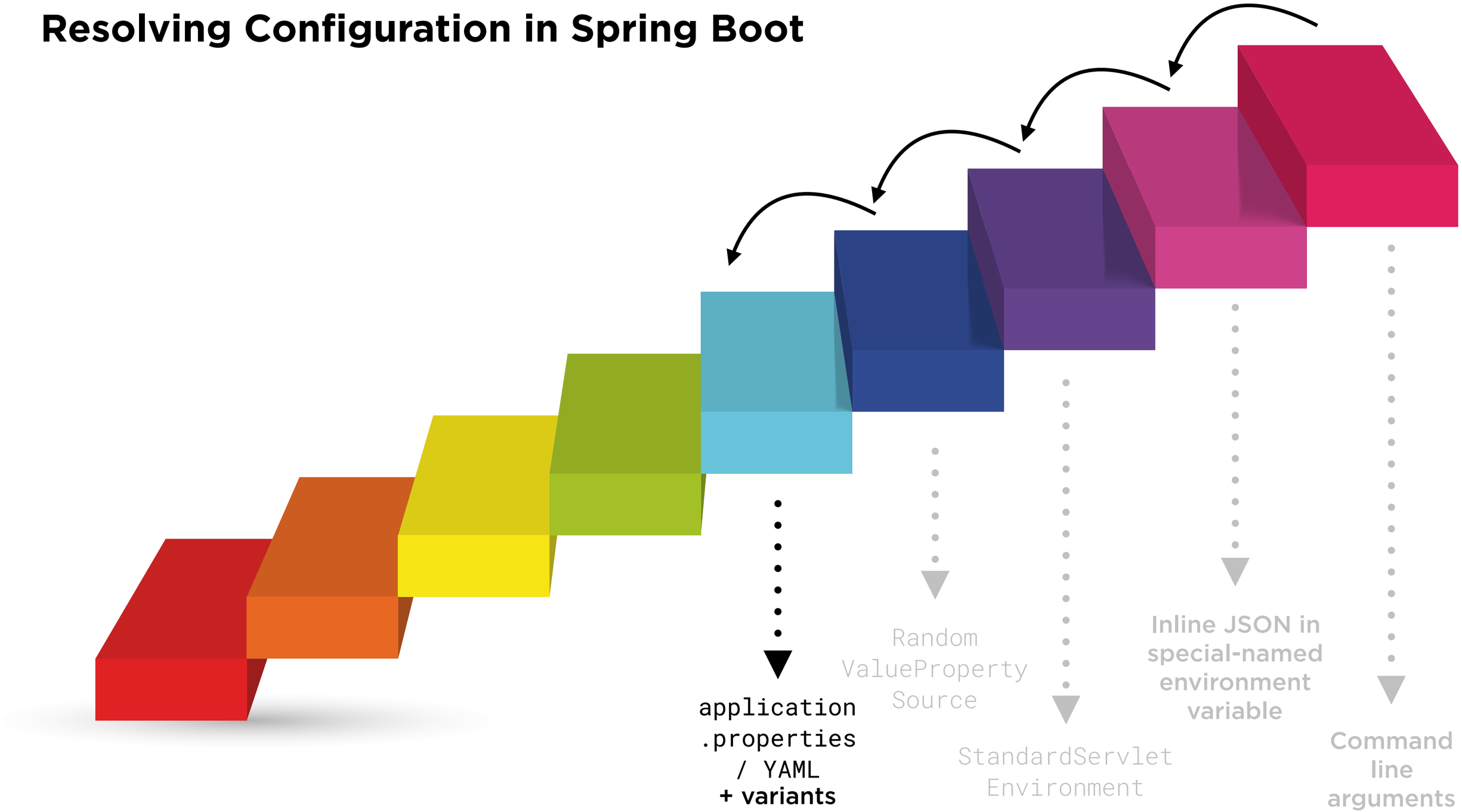
  B. int

  C. long

  D. int(<number>)

  E. int[<num1>,<num2>]

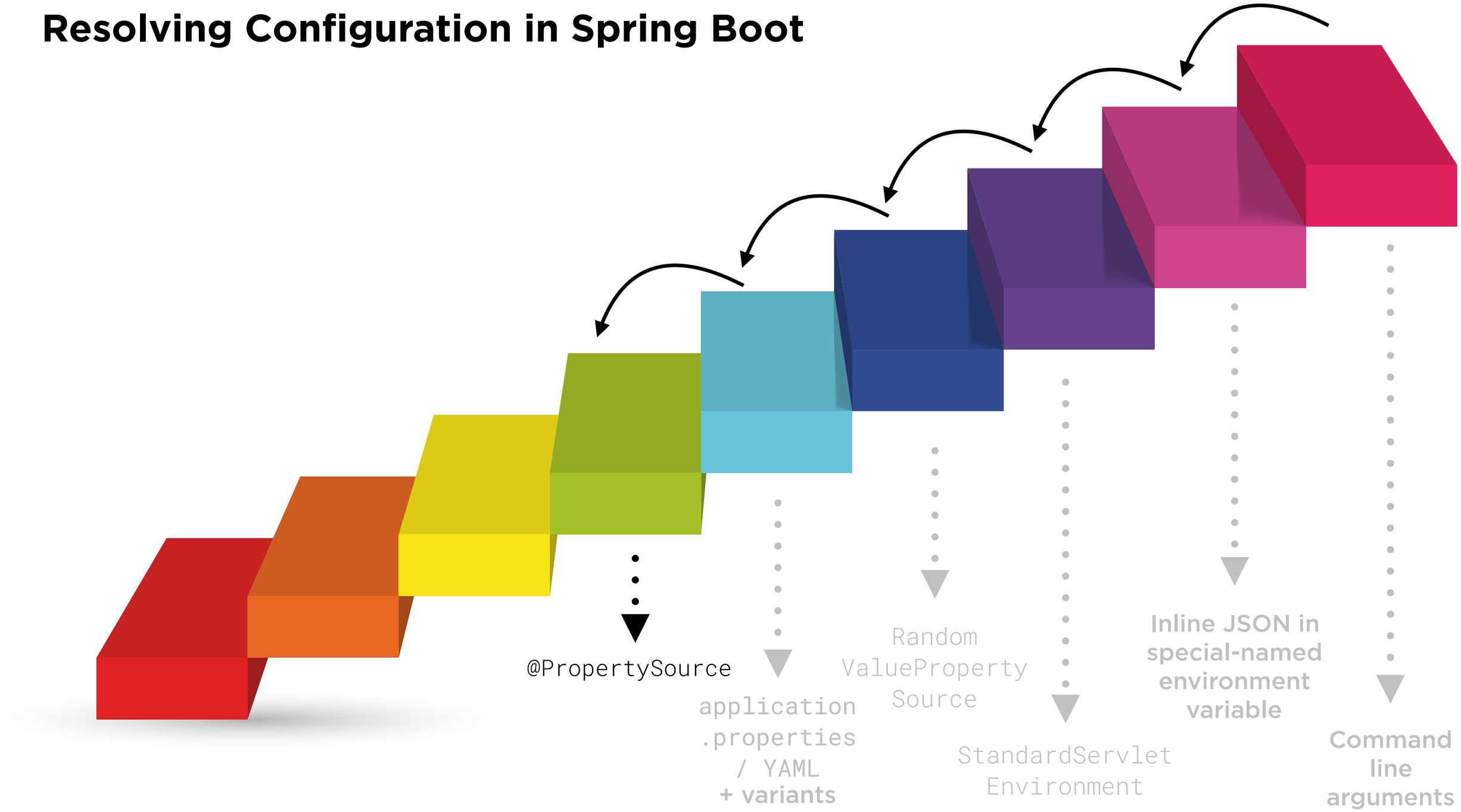
Resolving Configuration in Spring Boot



5.) `application.properties` / YAML + Variants

- **Look for profile-specific configuration 1st**
  - `application-{profile}.properties`
  - `application-{profile}.yml`
- **Look for generic configuration 2nd**
  - `application.properties / application.yml`
- **Check these locations**
  - `$CWD/config` **AND** `$CWD`
  - `classpath:/config` **AND** `classpath:`

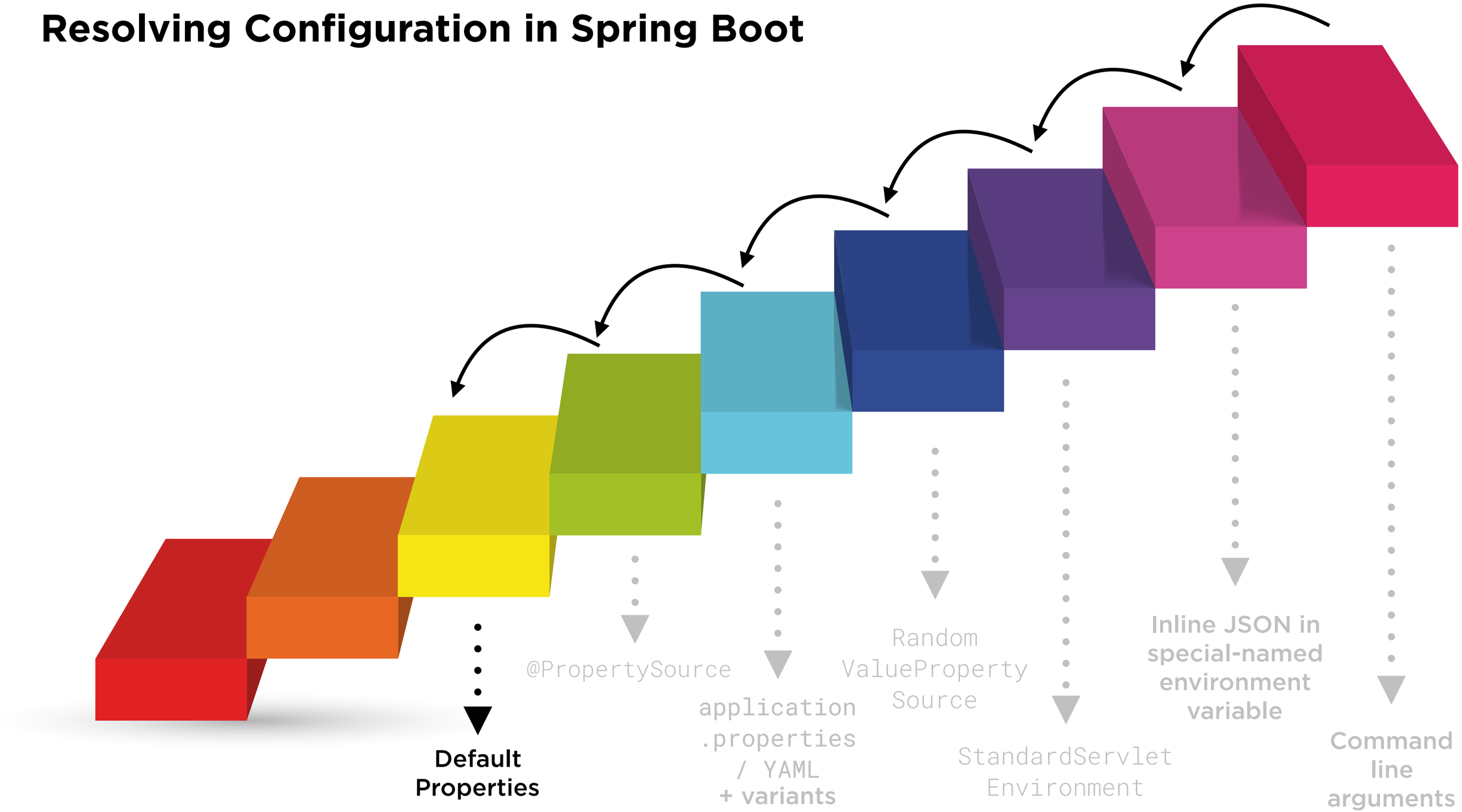
Resolving Configuration in Spring Boot



6.) @PropertySource

```
1 @SpringBootApplication
2 @PropertySource("/some/path/foo.properties")
3 public class MyApplication {
4     ...
5 }
```

Resolving Configuration in Spring Boot



7.) Default Properties

```
1 @SpringBootApplication
2 public class MyApplication {
3   public static void main(String args[])
4   {
5       SpringApplication.setDefaultProperties(...)
6   }
7 }
```

Recapping this Module

- **New configuration format: YAML**
- `@ConfigurationProperties`
- **Cascading resolution**

Deploying and Monitoring Your Spring Boot Apps in the Cloud

”Cloudier”

- Monitoring your apps in the cloud
  - Spring Boot provided endpoints
  - Writing custom health checks
- Deploy your Spring Boot apps to the cloud using Docker

Monitoring with Spring Boot

Introducing Spring Boot Actuator

- Production ready **monitoring** and **management features** out of the box
  - Health, autoconfig report, beans, etc
- HTTP or JMX
  - Feed into Nagios / Zabbix / New Relic
- Easy to add your own

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

pom.xml

Adding Spring Boot Actuator to Your Project

Builtin Production Ready Endpoints

`/autoconfig` **for report**

`/beans` **for all beans**

`/configprops` **for all config**

`/dump` **for memory dump**

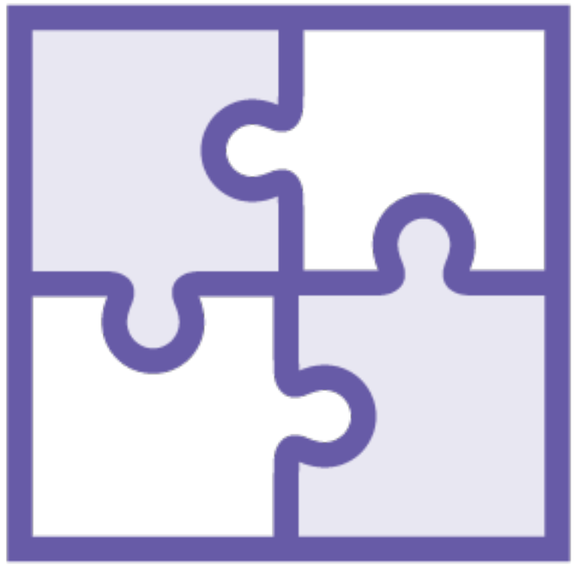
`/health` **to check application**

Many more ...

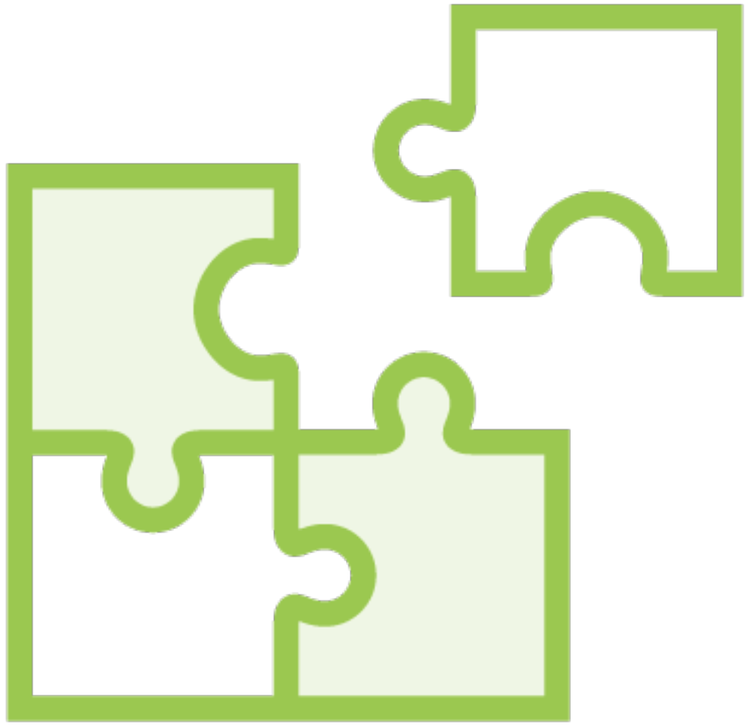
http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#production-ready

Is your application healthy?

Spring Boot `HealthIndicator`'s

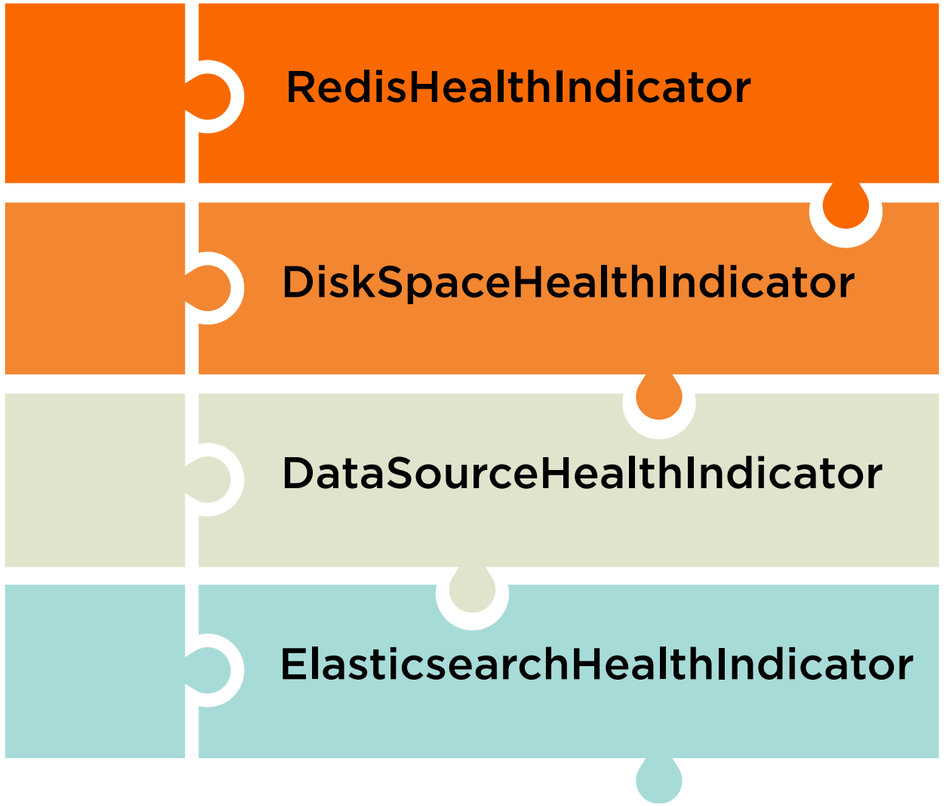


Autoconfigured



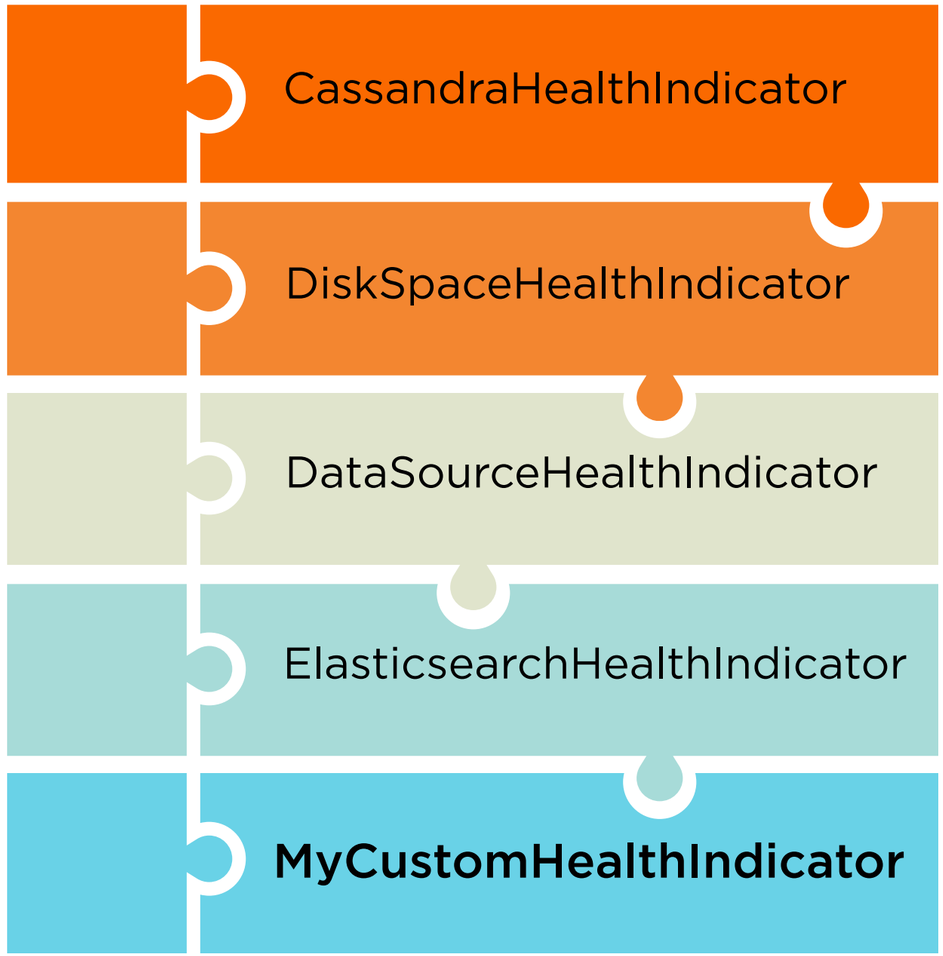
Custom

Autoconfigured `HealthIndicator`'s



http://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#_auto_configured_healthindicators

Custom `HealthIndicator`'s



```
1 @Component
2 public class MyCustomHealthIndicator implements HealthIndicator {
3
4
5
6
7
8 }
```

Defining Your Own `HealthIndicator`'s

Define a class that's annotated with `@Component` **and implements** `HealthIndicator`

```java
@Component
public class MyCustomHealthIndicator implements HealthIndicator {

    @Override
    public Health health() {
        ...
    }
}
```

Defining Your Own `HealthIndicator`'s

Implement the single `health()` method

```
// Condition failed
return Health.down().build();

// Condition failed with details (authentication required)
return Health.down().withDetail("someKey", "someValue").build();

// Condition is ok
return Health.up().build();

// Condition is unknown
return Health.unknown().build();
```

Defining Your Own `HealthIndicator`'s

Use the `Health` class's static builder methods to build `Health` object

Preparing Our Application for the Cloud

But wait, what if I don’t know anything about

?

What Is Docker?

- Virtualization management software for containers and images:
  - Build images
  - Deploy images into containers
  - Manage containers

What’s a container?

“A container is a stripped-to-basics version of a Linux operating system.”

Docker documentation

What’s an image?

The software you run in a container is called an ***image***

Why Docker?

Easy

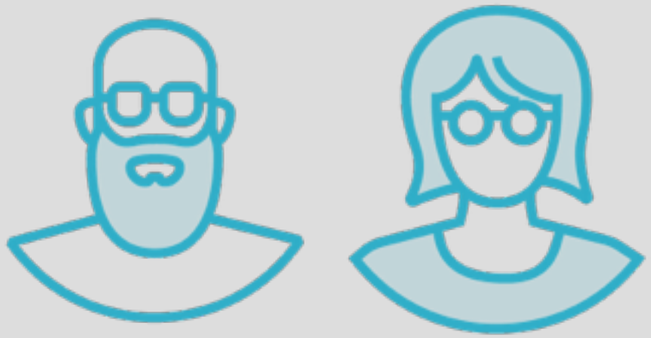
Lightweight

Cloud Agnostic

Scales

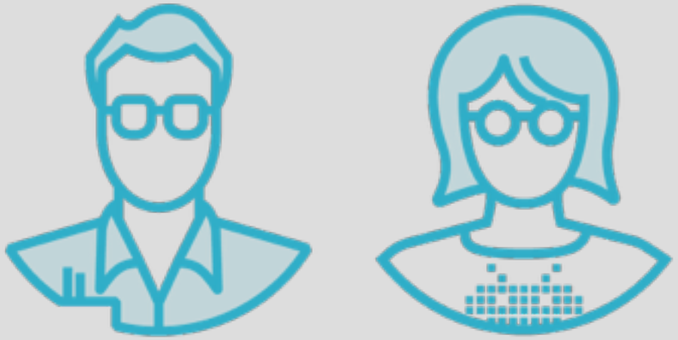
Installing Docker

Linux



https://docs.docker.com/engine/installation/linux/

Mac



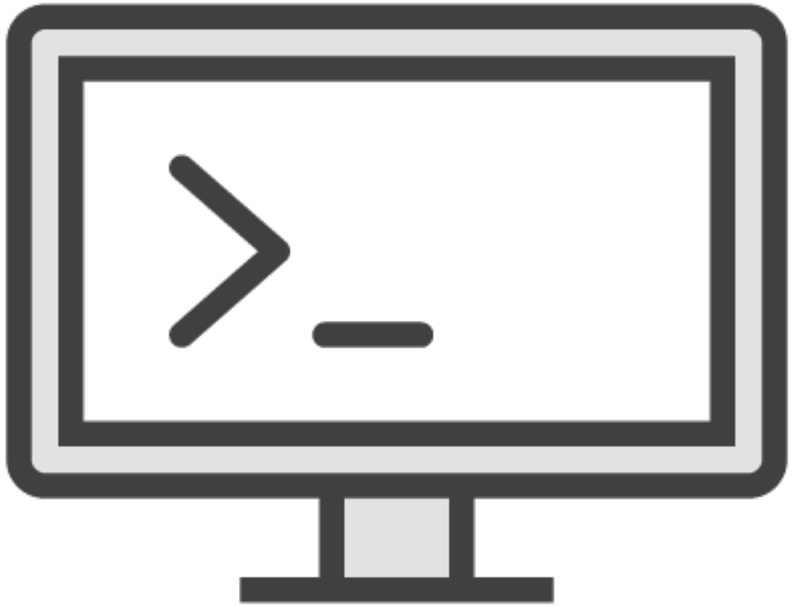
https://docs.docker.com/engine/installation/mac/

Windows

https://docs.docker.com/engine/installation/windows/

Docker Toolbox

Double click  **Docker Quickstart** to initialize Docker



Deploying Spring Boot to the Cloud

Amazon EC2 Container Service

- A container management service in the cloud
  - Supports Docker
- Highly scalable
  - Runs as a cluster
  - Can grow and shrink as needed
  - Load balancing (ELB)
- No additional charge

https://aws.amazon.com/

AWS Command Line Tool

https://aws.amazon.com/cli/

In Review...

- Spring Boot Actuator
- Spring Boot + Docker
- Deploying to the cloud (AWS)