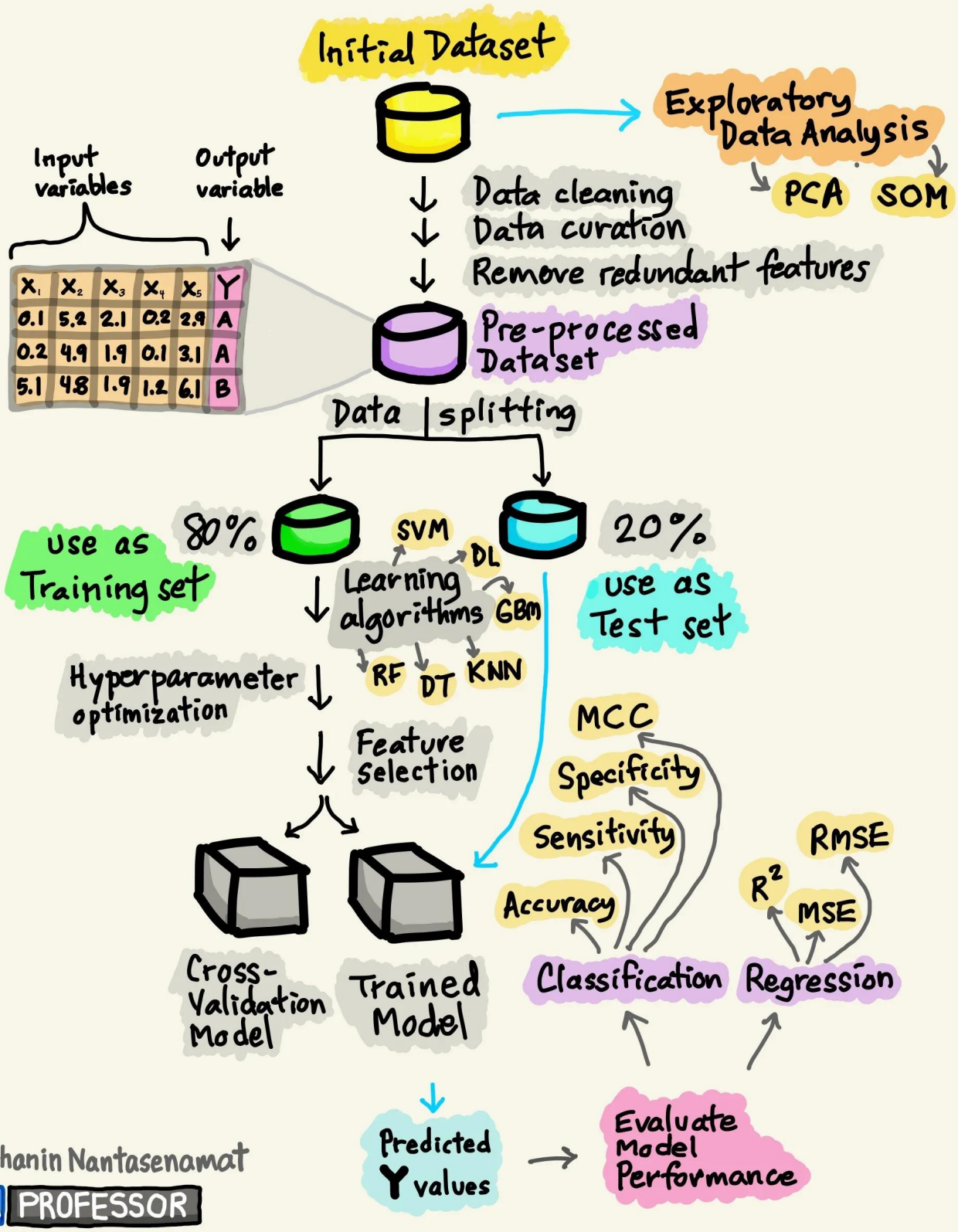
BUILDING THE MACHINE LEARNING MODEL



X_1	X_2	X_3	X_4	X_5	Y
0.1	5.2	2.1	0.2	2.9	A
0.2	4.9	1.9	0.1	3.1	A
5.1	4.8	1.9	1.2	6.1	B

By: Chanin Nantasenamat

DATA PROFESSOR

http://youtube.com/dataprofessor

January 1, 2020

Sources

- W3Schools.com
- DataQuest.io

SQL CHEATSHEET

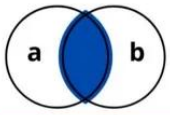
CONSIDER SUPPORTING ME

@AbzAaron

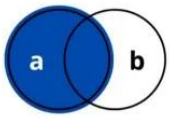
Commands / Clauses

SELECT	Select data from database
FROM	Specify table we're pulling from
WHERE	Filter query to match a condition
AS	Rename column or table with alias
JOIN	Combine rows from 2 or more tables
AND	Combine query conditions. All must be met
OR	Combine query conditions. One must be met
LIMIT	Limit rows returned. See also FETCH & TOP
IN	Specify multiple values when using WHERE
CASE	Return value on a specified condition
IS NULL	Return only rows with a NULL value
LIKE	Search for patterns in column
COMMIT	Write transaction to database
ROLLBACK	Undo a transaction block
ALTER TABLE	Add/Remove columns from table
UPDATE	Update table data
CREATE	Create TABLE, DATABASE, INDEX or VIEW
DELETE	Delete rows from table
INSERT	Add single row to table
DROP	Delete TABLE, DATABASE, or INDEX
GROUP BY	Group data into logical sets
ORDER BY	Set order of result. Use DESC to reverse order
HAVING	Same as WHERE but filters groups
COUNT	Count number of rows
SUM	Return sum of column
AVG	Return average of column
MIN	Return min value of column
MAX	Return max value of column

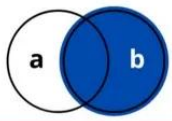
Joins



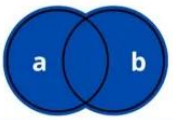
a INNER JOIN b



a LEFT JOIN b



a RIGHT JOIN b



a FULL OUTER JOIN b

Examples

Select all columns with filter applied

```
SELECT * FROM tbl
WHERE col > 5;
```

Select first 10 rows for two columns

```
SELECT col1, col2
FROM tbl LIMIT 10;
```

Select all columns with multiple filters

```
SELECT * FROM tbl
WHERE col1 > 5 OR col2 < 2;
```

Select all rows from col1 & col2 ordering by col1

```
SELECT col1, col2
FROM tbl ORDER BY 1;
```

Return count of rows in table

```
SELECT COUNT(*)
FROM tbl;
```

Return sum of col1

```
SELECT SUM(col1)
FROM tbl;
```

Return max value for col1

```
SELECT MAX(col1)
FROM tbl;
```

Compute summary stats by grouping col2

```
SELECT AVG(col1) FROM tbl
GROUP BY col2;
```

Combine data from 2 tables using left join

```
SELECT * FROM tbl1 AS t1 LEFT JOIN
tbl2 AS t2 ON t2.col1 = t1.col1;
```

Aggregate and filter result

```
SELECT col1,
       COUNT(*) AS total
FROM tbl
GROUP BY col1
HAVING COUNT(*) > 10;
```

Implementation of CASE statement

```
SELECT col1,
CASE
    WHEN col1 > 10 THEN 'more than 10'
    WHEN col1 < 10 THEN 'less than 10'
    ELSE '10'
END AS NewColumnName
FROM tbl;
```

Data Definition Language

CREATE

```
CREATE DATABASE MyDatabase;

CREATE TABLE MyTable (
    id int,
    name varchar(10));

CREATE INDEX IndexName
ON TableName(col1);
```

ALTER

```
ALTER TABLE MyTable
DROP COLUMN col5;

ALTER TABLE MyTable
ADD col5 int;
```

DROP

```
DROP DATABASE MyDatabase;
DROP TABLE MyTable;
```

Data Manipulation Language

UPDATE

```
UPDATE MyTable
SET col1 = 56
WHERE col2 = 'something';
```

INSERT

```
INSERT INTO MyTable (col1, col2)
VALUES ('value1', 'value2');
```

DELETE

```
DELETE FROM MyTable
WHERE col1 = 'something';
```

SELECT

```
SELECT col1, col2
FROM MyTable;
```

Order Of Execution

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT

JVM Options Cheat Sheet

JRebel | XRebel

STANDARD OPTIONS

`$ java`
List all standard options.

`-Dblog=JRebelBlog`
Sets a 'blog' system property to 'JRebelBlog'. Retrieve/set it during runtime like this:

```
System.getProperty("blog");
//JRebelBlog

System.setProperty("blog", "JR");
```

`-javaagent:/path/to/agent.jar`
Loads the java agent in agent.jar.

`-agentpath:pathname`
Loads the native agent library specified by the absolute path name.

`-verbose:[class/gc/jni]`
Displays information about each loaded class/ gc event/JNI activity.

NON-STANDARD OPTIONS

`$ java -X`
List all non-standard options.

`-Xint`
Runs the application in interpreted-only mode.

`-Xbootclasspath:path`
Path and archive list of boot class files.

`-Xloggc:filename`
Log verbose GC events to filename.

`-Xms1g`
Set the initial size (in bytes) of the heap.

`-Xmx8g`
Specifies the max size (in bytes) of the heap.

`-Xnoclassgc`
Disables class garbage collection.

`-Xprof`
Profiles the running program.

ADVANCED OPTIONS

BEHAVIOR

`-XX:+UseConcMarkSweepGC`
Enables CMS garbage collection.

`-XX:+UseParallelGC`
Enables parallel garbage collection.

`-XX:+UseSerialGC`
Enables serial garbage collection.

`-XX:+UseG1GC`
Enables G1GC garbage collection.

DEBUGGING

`-XX:ErrorFile=file.log`
Save the error data to file.log.

`-XX:+HeapDumpOnOutOfMemoryError`
Enables heap dump when `OutOfMemoryError` is thrown.

`-Xlog:gc`
Enables printing messages during garbage collection.

`-XX:+TraceClassLoading`
Enables Trace loading of classes.

`-XX:+PrintClassHistogram`
Enables printing of a class instance histogram after a `Control+C` event `(SIGTERM)`.

PERFORMANCE

`-XX:ThreadStackSize=256k`
Sets Thread Stack Size (in bytes).
(Same as -Xss256k)

`-XX:G1HeapRegionSize=4m`
Sets the sub-division size of G1 heap (in bytes).

`-XX:MaxGCPauseMillis=n`
Sets a target for the maximum GC pause time.

`-XX:MaxNewSize=256m`
Max size of new generation (in bytes).

`-XX:OnError="<cmd args>"`
Run user-defined commands on fatal error.

spring Annotations Cheat Sheet

JRebel

Spring Boot and Web Annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses @Configuration, @EnableAutoConfiguration and @ComponentScan.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

T **@Controller** - marks the class as web controller, capable of handling the requests.

T **@RestController** - a convenience annotation of a @Controller and @ResponseBody.

M T **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.

Spring Framework Annotations

Spring uses dependancy injection to configure and bind your appliation together.

T **@Configuration** - used to mark a class as a source of the bean definitions.

T **@ComponentScan** - makes Spring scan the packages configured with it for the @Configuration classes.

T **@Import** - loads additional configuration. This one works even when you specify the beans in an XML file.

T **@Component** - turns the class into a Spring bean at the auto-scan time.

T **@Service** - tells Spring that it's safe to manage @Components with more freedom than regular components.

C F M **@Autowired** - wires the application parts together, on the fields, constructors, or methods in a component.

M **@Bean** - specifies a returned bean to be managed by Spring context. The returned bean has the same name as the factory method.

M **@Lookup** - tells Spring to return an instance of the method's return type when we invoke it.

T M **@Primary** - gives higher preference to a bean when there are multiple beans of the same type.

C F M **@Required** - shows that the setter method must be configured to be dependency-injected with a value at configuration time.

C F M **@Value** - used to assign values into fields in Spring-managed beans. It's compatible with the constructor, setter, and field injection.

T M **@DependsOn** - makes Spring initialize other beans before the annotated one.

T M **@Lazy** - makes beans to initialize lazily. @Lazy annotation may be used on any class directly or indirectly annotated with @Component or on methods annotated with @Bean.

T M **@Scope** - used to define the scope of a @Component class or a @Bean definition and can be either singleton, prototype, request, session, globalSession, or custom scope.

T **@Profile** - adds beans to the application only when that profile is active.

Legend: T - Class F - Field Annotation C - Constructor Annotation M - Method P - Parameter

docker Cheat Sheet

JRebel | XRebel

GLOSSARY

Layer
A set of read-only files to provision the system.

Image
A read-only layer that is the base of your container. Might have a parent image.

Container
A runnable instance of the image.

Registery / Hub
Central place where images live.

Docker machine
A VM to run Docker containers (Linux does this natively).

Docker compose
A utility to run multiple containers as a system.

USEFUL ONE-LINERS

Download an image
```
docker pull image_name
```

Start and stop the container
```
docker [start|stop] container_name
```

Create and start container, run command
```
docker run -ti --name container_name
    image_name command
```

Create and start container, run command, destroy container
```
docker run --rm -ti image_name command
```

Example filesystem and port mappings
```
docker run -it --rm -p 8080:8080 -v
/path/to/agent.jar:/agent.jar -e
JAVA_OPTS="-javaagent:/agent.jar"
tomcat:8.0.29-jre8
```

DOCKER CLEANUP COMMANDS

Kill all running containers
```
docker kill $(docker ps -q)
```

Delete dangling images
```
docker rmi $(docker images -q -f
    dangling=true)
```

Remove all stopped containers
```
docker rm $(docker ps -a -q)
```

DOCKER MACHINE COMMANDS

Use docker-machine to run the containers

Start a machine
```
docker-machine start machine_name
```

Configure docker to use a specific machine
```
eval "$(docker-machine env machine_name)"
```

DOCKER COMPOSE SYNTAX

docker-compose.yml file example
```
version: "2"
services:
web:
  container_name: "web"
  image: java:8 # image name
  # command to run
  command: java -jar /app/app.jar
  ports: # map ports to the host
    - "4567:4567"
  volumes: # map filesystem to the host
    - ./myapp.jar:/app/app.jar
mongo: # container name
  image: mongo # image name
```

Create and start containers
```
docker-compose up
```

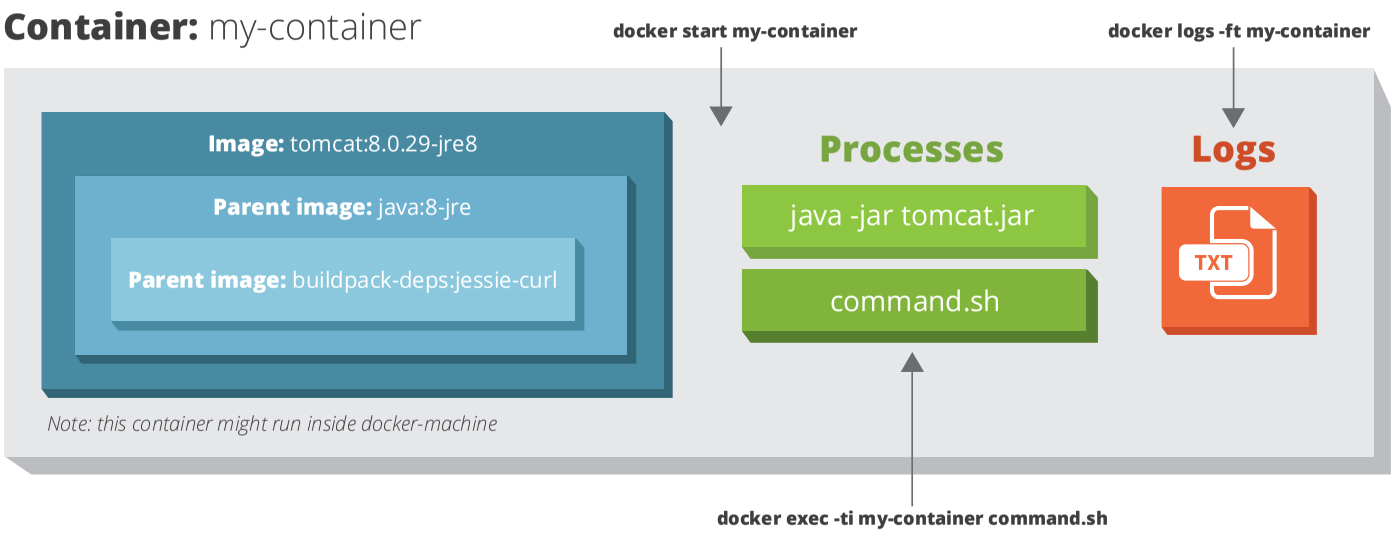
INTERACTING WITH A CONTAINER

Run a command in the container
```
docker exec -ti container_name command.sh
```

Follow the container logs
```
docker logs -ft container_name
```

Save a running container as an image
```
docker commit -m "commit message" -a "author"
    container_name username/image_name:tag
```

Container: my-container



Java 8 Best Practices Cheat Sheet

Default methods

Evolve interfaces & create traits

```
//Default methods in interfaces
@FunctionalInterface
interface Utilities {
  default Consumer<Runnable> m() {
    return (r) -> r.run();
  }
  // default methods, still functional
  Object function(Object o);
}
class A implements Utilities { // implement
  public Object function(Object o) {
    return new Object();
  }
  {
    // call a default method
    Consumer<Runnable> n = new A().m();
  }
}
```

Lambdas

Syntax:
(parameters) -> expression
(parameters) -> { statements; }

```
// takes a Long, returns a String
Function<Long, String> f = (l) -> l.toString();
// takes nothing gives you Threads
Supplier<Thread> s = Thread::currentThread;
// takes a string as the parameter
Consumer<String> c = System.out::println;

// use them with streams
new ArrayList<String>().stream().
// peek: debug streams without changes
peek(e -> System.out.println(e)).
// map: convert every element into something
map(e -> e.hashCode()).
// filter: pass some elements through
filter (hc -> (hc % 2) == 0).
// collect all values from the stream
collect(Collectors.toCollection(TreeSet::new))
```

java.util. Optional

A container for possible null values

```
// Create an optional
Optional<String> optional =
Optional.ofNullable(a);
// process the optional
optional.map(s -> "RebelLabs:" + s);
// map a function that returns Optional
optional.flatMap(s -> Optional.ofNullable(s));

// run if the value is there
optional.ifPresent(System.out::println);
// get the value or throw an exception
optional.get();

// return the value or the given value
optional.orElse("Hello world!");
// return empty Optional if not satisfied
optional.filter(s -> s.startsWith("RebelLabs"));
```

Rules of Thumb

Traits: 1 default method per interface
Don't enhance functional interfaces
Only conservative implementations

Expressions over statements
Refactor to use method references
Chain lambdas rather than growing them

Fields - use plain objects
Method parameters, use plain objects
Return values - use Optional
Use *orElse()* instead of *get()*

Java Generics cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Basics

Generics don't exist at runtime!

```
class Pair<T1, T2> { /* ... */ }
```
-- the type parameter section, in angle brackets, specifies type variables.

Type parameters are substituted when objects are instantiated.

```
Pair<String, Long> p1 = new
  Pair<String, Long> ("RL", 43L);
```

Avoid verbosity with the diamond operator:

```
Pair<String, Long> p1 =
  new Pair<>("RL", 43L);
```

Wildcards

`Collection<Object>` - heterogenous, any object goes in.
`Collection<?>` - homogenous collection of arbitrary type.

Avoid using wildcards in return types!

Intersection types

```
<T extends Object &
  Comparable<? super T>> T
max(Collection<? extends T> coll)
```

The return type here is ***Object***!

Compiler generates the bytecode for the most general method only.

Producer Extends Consumer Super (PECS)

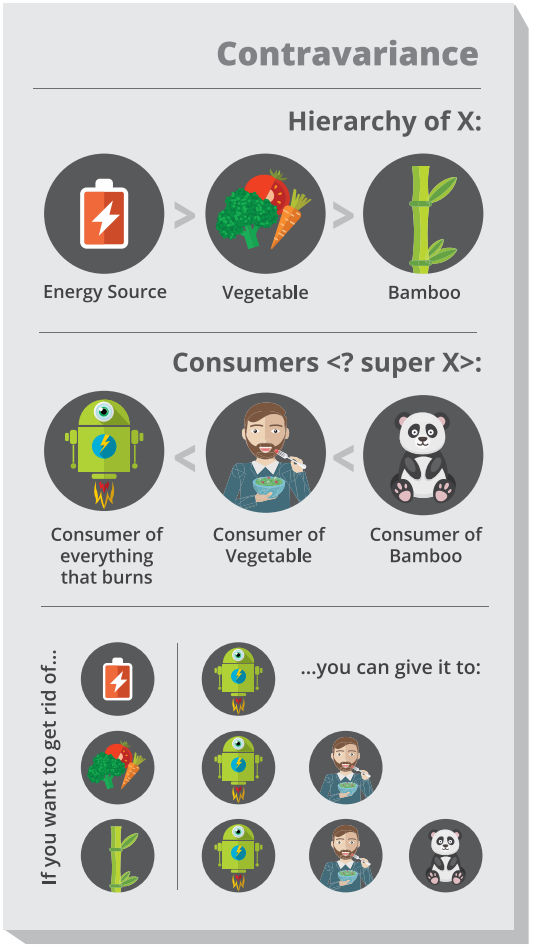
```
Collections.copy(List<? super T> dest, List<? extends T> src)
```

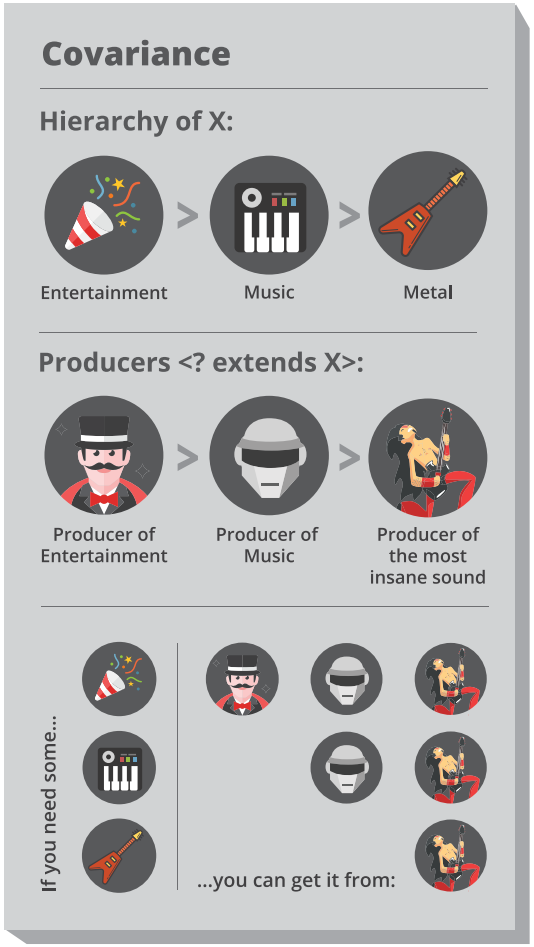
src -- contains elements of type T or its subtypes.

dest -- accepts elements, so defined to use T or its supertypes.

*Consumers are **contravariant** (use super).*

*Producers are **covariant** (use extends).*





Method Overloading

```
String f(Object s) {
  return "object";
}
String f(String s) {
  return "string";
}
<T> String generic(T t) {
  return f(t);
}
```

If called **generic("string")** returns "object".

Recursive generics

Recursive generics add constraints to your type variables. This helps the compiler to better understand your types and API.

```
interface Cloneable<T extends
Cloneable<T>> {
  T clone();
}
```

Now `cloneable.clone().clone()` will compile.

Covariance

```
List<Number> ⊁ ArrayList<Integer>
```

Collections are not covariant!

BROUGHT TO YOU BY JRebel

THE JAVA LANGUAGE CHEAT SHEET

Primitive Types:

INTEGER: byte(8bit),short(16bit),int(32bit),
long(64bit),**DECIM:**float(32bit),double(64bit)
,**OTHER:** boolean(1bit), char (Unicode)
HEX:0x1AF,**BINARY:**0b00101,**LONG:**8888888888888**L**
CHAR EXAMPLES: 'a','\n','\t','\'','\\','\"'

Primitive Operators

Assignment Operator: = (ex: int a=5,b=3;)
Binary Operators (two arguments): + - * / %
Unary Operators: + - ++ --
Boolean Not Operator (Unary): !
Boolean Binary: == != > >= < <=
Boolean Binary Only: && ||
Bitwise Operators: ~ & ^ | << >> >>>
Ternary Operator: bool?valtrue:valfalse;

Casting, Conversion

```
int x = (int)5.5; //works for numeric types
int x = Integer.parseInt("123");
float y = Float.parseFloat("1.5");
int x = Integer.parseInt("7A",16); //fromHex
String hex = Integer.toString(99,16);//toHex
//Previous lines work w/ binary, other bases
```

java.util.Scanner, input, output

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt(); //stops at whitespace
String line = sc.nextLine(); //whole line
System.out.println("bla"); //stdout
System.err.print("bla"); //stderr,no newline
```

java.lang.Number types

```
Integer x = 5; double y = x.doubleValue();
double y = (double)x.intValue();
//Many other methods for Long, Double, etc
```

java.lang.String Methods

```
//Operator +, e.g. "fat"+"cat" -> "fatcat"
boolean equals(String other);
int length();
char charAt(int i);
String substring(int i, int j); //j not incl
boolean contains(String sub);
boolean startsWith(String pre);
boolean endsWith(String post);
int indexOf(String p); //-1 if not found
int indexOf(String p, int i); //start at i
int compareTo(String t);
//"a".compareTo("b") -> -1
String replaceAll(String str, String find);
String[] split(String delim);
```

StringBuffer, StringBuilder

StringBuffer is synchronized StringBuilder
(Use StringBuilder unless multithreaded)
Use the .apend(xyz) methods to concat
toString() converts back to String

java.lang.Math

```
Math.abs(NUM),Math.ceil(NUM),Math.floor(NUM)
,Math.log(NUM),Math.max(A,B),Math.min(C,D),
Math.pow(A,B),Math.round(A),Math.random()
```

IF STATEMENTS:

```
if( boolean_value ) { STATEMENTS }
else if( bool )     { STATEMENTS }
else if( ..etc )    { STATEMENTS }
else                { STATEMENTS }
//curly brackets optional if one line
```

LOOPS:

```
while( bool )         { STATEMENTS }
for(INIT;BOOL;UPDATE) { STATEMENTS }
//1INIT 2BOOL 3STATEMENTS 4UPDATE 5->Step2
do{ STATEMENTS }while( bool );
//do loops run at least once before checking
break;    //ends enclosing loop (exit loop)
continue; //jumps to bottom of loop
```

ARRAYS:

```
int[] x = new int[10]; //ten zeros
int[][] x = new int[5][5]; //5 by 5 matrix
int[] x = {1,2,3,4};
x.length; //int expression length of array
int[][] x = {{1,2},{3,4,5}}; //ragged array
String[] y = new String[10]; //10 nulls
//Note that object types are null by default

//loop through array:
for(int i=0;i<arrayname.length;i++) {
  //use arrayname[i];
}

//for-each loop through array
int[] x = {10,20,30,40};
for(int v : x) {
  //v cycles between 10,20,30,40
}

//Loop through ragged arrays:
for(int i=0;i<x.length;i++)
  for(int j=0;j<x[i].length;j++) {
     //CODE HERE
  }

//Note, multi-dim arrays can have nulls
//in many places, especially object arrays:
Integer[][] x = {{1,2},{3,null},null};
```

FUNCTIONS / METHODS:

```
Static Declarations:
public static int functionname( … )
private static double functionname( … )
static void functionname( … )
Instance Declarations:
public void functionname( … )
private int functionname( … )
Arguments, Return Statement:
int myfunc(int arg0, String arg1) {
  return 5; //type matches int myfunc
}
//Non-void methods must return before ending
//Recursive functions should have an if
//statement base-case that returns at once
```

CLASS/OBJECT TYPES:

```
INSTANTIATION:
public class Ball {//only 1 public per file
  //STATIC FIELDS/METHODS
  private static int numBalls = 0;
  public static int getNumBalls() {
    return numBalls;
  }
  public static final int BALLRADIUS = 5;

  //INSTANCE FIELDS
  private int x, y, vx, vy;
  public boolean randomPos = false;

  //CONSTRUCTORS
  public Ball(int x, int y, int vx, int vy)
  {
    this.x = x;
    this.y = y;
    this.vx = vx;
    this.vy = vy;
    numBalls++;
  }
  Ball() {
    x = Math.random()*100;
    y = Math.random()*200;
    randomPos = true;
  }

  //INSTANCE METHODS
  public int getX(){ return x; }
  public int getY(){ return y; }
  public int getVX(){ return vx; }
  public int getVY(){ return vy; }
  public void move(){ x+=vx; y+=vy; }
  public boolean touching(Ball other) {
    float dx = x-other.x;
    float dy = y-other.y;
    float rr = BALLRADIUS;
    return Math.sqrt(dx*dx+dy*dy)<rr;
  }

}

//Example Usage:
public static void main(String[] args) {
  Ball x = new Ball(5,10,2,2);
  Ball y = new Ball();
  List<Ball> balls = new ArrayList<Ball>();
  balls.add(x); balls.add(y);
  for(Ball b : balls) {
    for(Ball o : balls) {
      if(b != o) { //compares references
        boolean touch = b.touching(o);
      }
    }
  }
}
```

POLYMORPHISM:

Single Inheritance with "extends"

```
class A{ }
class B extends A{ }
abstract class C { }
class D extends C { }
class E extends D
```

Abstract methods

```
abstract class F {
  abstract int bla();
}
class G extends F {
  int bla() {  //required method
   return 5;
  }
}
```

Multiple Inheritance of interfaces with "implements" (fields not inherited)

```
interface H {
  void methodA();
  boolean methodB(int arg);
}
interface I extends H{
  void methodC();
}
interface K {}
class J extends F implements I, K {
  int bla() { return 5; } //required from F
  void methodA(){} //required from H
  boolean methodB(int a) { //req from A
    return 1;
  }
  void methodC(){} //required from I
}
```

Type inference:

```
A x = new B(); //OK
B y = new A(); //Not OK
C z = new C(); //Cannot instantiate abstract
//Method calls care about right hand type
(the instantiated object)
//Compiler checks depend on left hand type
```

GENERICS:

```
class MyClass<T> {
  T value;
  T getValue() { return value; }
}
class ExampleTwo<A,B> {
  A x;
  B y;
}
class ExampleThree<A extends List<B>,B> {
  A list;
  B head;
}
//Note the extends keyword here applies as
well to interfaces, so A can be an interface
that extends List<B>
```

JAVA COLLECTIONS:

List<T>: Similar to arrays

```
ArrayList<T>: Slow insert into middle
//ArrayList has fast random access
LinkedList<T>: slow random access
//LinkedList fast as queue/stack
Stack: Removes and adds from end
```

List Usage:

```
boolean add(T e);
void clear(); //empties
boolean contains(Object o);
T get(int index);
T remove(int index);
boolean remove(Object o);
//remove uses comparator
T set(int index, E val);
Int size();
```

List Traversal:

```
for(int i=0i<x.size();i++) {
       //use x.get(i);
}

//Assuming List<T>:
for(T e : x) {
       //use e
}
```

Queue<T>: Remove end, Insert beginning

```
LinkedList implements Queue
```

Queue Usage:

```
T element(); // does not remove
boolean offer(T o); //adds
T peek(); //pike element
T poll(); //removes
T remove(); //like poll
Traversal: for(T e : x) {}
```

Set<T>: uses Comparable<T> for uniqueness

```
TreeSet<T>, items are sorted
HashSet<T>, not sorted, no order
LinkedHashSet<T>, ordered by insert
Usage like list: add, remove, size
Traversal: for(T e : x) {}
```

Map<K,V>: Pairs where keys are unique

```
HashMap<K,V>, no order
LinkedHashMap<K,V> ordered by insert
TreeMap<K,V> sorted by keys

V get(K key);
Set<K> keySet(); //set of keys
V put(K key, V value);
V remove(K key);
Int size();
Collection<V> values(); //all values

Traversal: for-each w/ keyset/values
```

java.util.PriorityQueue<T>

A queue that is always automatically sorted using the comparable function of an object

```
public static void main(String[] args) {
  Comparator<String> cmp= new LenCmp();
  PriorityQueue<String> queue =
     new PriorityQueue<String>(10, cmp);
  queue.add("short");
  queue.add("very long indeed");
  queue.add("medium");
  while (queue.size() != 0)
    System.out.println(queue.remove());
}
class LenCmp implements Comparator<String> {
  public int compare(String x, String y){
    return x.length() - y.length();
  }
}
```

java.util.Collections algorithms

Sort Example:

```
//Assuming List<T> x
Collections.sort(x); //sorts with comparator
```

Sort Using Comparator:

```
Collections.sort(x, new Comparator<T>{
  public int compareTo(T a, T b) {
    //calculate which is first
    //return -1, 0, or 1 for order:
    return someint;
  }
}
```

Example of two dimensional array sort:

```
public static void main(final String[] a){
   final String[][] data = new String[][] {
     new String[] { "20090725", "A" },
     new String[] { "20090726", "B" },
     new String[] { "20090727", "C" },
     new String[] { "20090728", "D" } };
     Arrays.sort(data,
       new Comparator<String[]>() {
        public int compare(final String[]
entry1, final String[] entry2) {
          final String time1 = entry1[0];
          final String time2 = entry2[0];
          return time1.compareTo(time2);
        }
     });

     for (final String[] s : data) {
       System.out.println(s[0]+" "+s[1]);
     }
   }
}
```

More collections static methods:

```
Collections.max( … ); //returns maximum
Collections.min( … ); //returns maximum
Collections.copy( A, B); //A list into B
Collections.reverse( A ); //if A is list
```

JUnit cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Assertions and assumptions

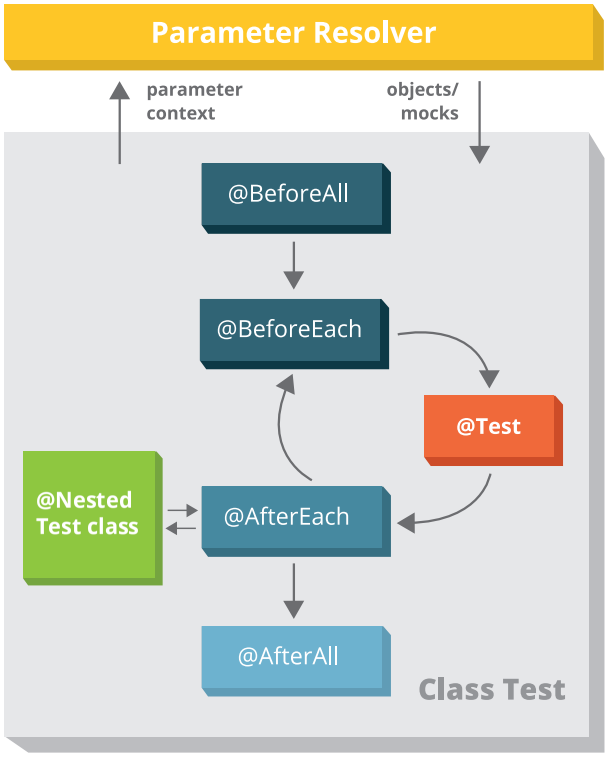
Use assertions to verify the behavior under test:

```
Assertions.assertEquals(Object expected,
Object actual, Supplier<String> message)

// Group assertions are run all together and
reported together.
Assertions.assertAll("heading",
  () -> assertTrue(true),
  () -> assertEquals("expected",
            objectUnderTest.getSomething());

// To check for an exception:
expectThrows(NullPointerException.class,
  () -> { ((Object) null).getClass();});
```

Lifecycle of standard tests



Parameter resolution

ParameterResolver - extension interface to provide parameters

```
public class MyInfoResolver implements ParameterResolver {
  public Object resolve(ParameterContext paramCtx,
                        ExtensionContext extCtx) {
    return new MyInfo();
  }
}
```

Extend your tests with your parameter resolver.

```
@ExtendWith(MyInfoResolver.class)
class MyInfoTest { … }
```

Useful code snippets

```
@TestFactory
Stream<DynamicTest> dynamicTests(MyContext ctx) {
// Generates tests for every line in the file
 return Files.lines(ctx.testDataFilePath).map(l ->
dynamicTest("Test:" + l, () -> assertTrue(runTest(l)));
}

@ExtendWith({ MockitoExtension.class,
DataParameterProvider.class })
class Tests {
  ArrayList<String> list;

  @BeforeEach
  void init() { /* init code */ }

  @Test
  @DisplayName("Add elements to ArrayList")
  void addAllToEmpty(Data dep) {
    list.addAll(dep.getAll());
    assertAll("sizes",
      () -> assertEqual(dep.size(), list.size(),
              () -> "Unexpected size:" + instance),
      () -> assertEqual(dep.getFirst(), list.get(0),
              () -> "Wrong first element" + instance));
  }
}
```

Useful annotations

@Test - marks a test method
@TestFactory - method to create test cases at Runtime
@DisplayName - make reports readable with proper test names
@BeforeAll/@BeforeEach - lifecycle methods executed prior testing
@AfterAll/AfterEach - lifecycle methods for cleanup
@Tag - declare tags to separate tests into suites
@Disabled - make JUnit skip this test.

Use **@Nested** on an inner class to control the order of tests.

Use **@ExtendWith()** to enhance the execution: provide mock parameter resolvers and specify conditional execution.

Use the lifecycle and **@Test** annotations on the default methods in interfaces to define contracts:

```
@Test
interface HashCodeContract<T> {
  <T> getValue();
  <T> getAnotherValue();
  @Test
  void hashCodeConsistent() {
    assertEquals(getValue().hashCode(),
                 getAnotherValue().hashCode(),
                 "Hashes differ");
}
```

"Never trust a test you haven't seen fail."

— Colin Vipurs

KUBERNETES CHEAT SHEET

KUBERNETES

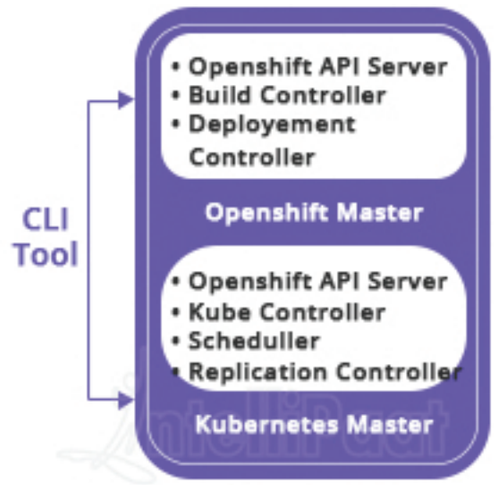
• It is an open source platform for automating deployment and scaling of containers across clusters of hosts providing container centric infrastructure.

• It is a container orchestrator and can run Linux containers:

• Launch container.

• Maintain and monitor container site.

• Performs container-oriented networking



Master(s)
Manage,Build,Deploy & Operate

NODE(s)
Run Containers and registries

Key Concepts

Now let's discuss the key points of this architecture.

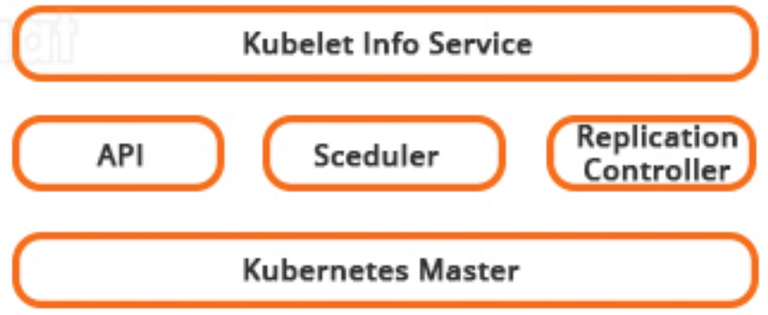
• Pod: These are the group of containers.
• Labels: These are used to identify the pods.
• Kubelet: They are container agents, responsible for maintaining the set of pods.
• Proxy: They are the Load balancer for pods, helping in distributing tasks across the pods.
• ETCD: A Metadata service.

• Cadvisor: For resource usage and performance stats.
• Replication controller: It manages pod replication.
• Scheduler: Used for pod scheduling in worker nodes.
• API server: Kubernetes API server.

Now let's understand the role Master and Node play in the Kubernetes Architecture.

Master

• It is responsible for maintaining the desired state for the cluster you are working on.
• "Master" indicates a set of processes that are used to manage the cluster.
• Contains info, API, scheduler, replication controllers, and master.

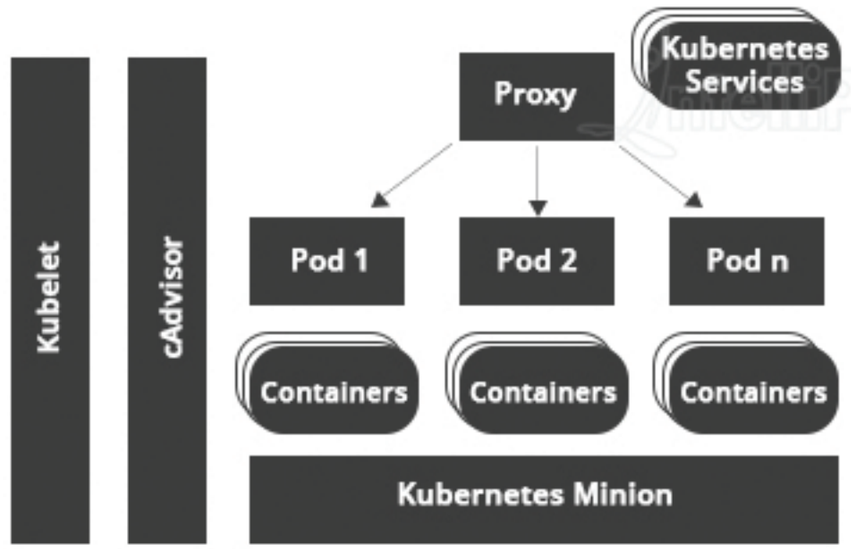


Worker Nodes / Minions

• Also called as a minion. It contains the services necessary to run the pods that are managed by the master.

• Some services include: container runtime, Kubelet, kube-proxy.

• Contains: Kubelet, cAdvisor, services, pods and containers.



Features

• Automated scheduling- provides an advanced scheduler that helps launch container on cluster nodes
• Self healing- reschedule, replace and restart dead containers.
• Automated rollouts and rollbacks- supports rollback for systems incase of a failure. Enables rollout and rollback for the desired state.
• Horizontal scaling- can scale up and down the app as per required. Can also be automated wrt CPU usage.
• Service discovery and load balancing- uses unique ip and dns name to containers. This helps identify them across different containers.

• Pods and Container Introspection

COMMANDS	FUNCTION
Kubectl get pods	Lists all current pods
Kubectl describe pod<name>	Describes the pod names
Kubectl get rc	List all replication controllers
Kubectl get rc --namespace="namespace"	Lists all current pods
Kubectl describe rc <name>	Shows the replication controller name
Kubectl get cvc	Lists the services
Kubectl describe svc <name>	Shows the service name
Kubectl delete pod <name>	Deletes the pod
Kubectl get nodes -w	Watch nodes continuously

• Debugging

COMMANDS	FUNCTION
Execute command on service by selecting container.	Kubectl exec<service><commands>[-c< $container>]
Get logs from service for a container	Kubectl logs -f<name>>[-c< $container>]
Watch the kubelet logs	Watch -n 2 cat/var/log/kublet.log
Show metrics for node	Kubectl top node
Show metrics for pods	Kubectl top pod

• Objects

All	Clusterroleblindings	FUNCTION
cm= conf gmaps c	controllerrevisions	crd=custom resource definition
Cronjobs	cs=component status	csr= certificate signing requests
Deploy=deployments	ds= daemon sets	ep=end points
ev= events	hpa= autoscaling	ing= ingress
jobs	limits=limit ranges	Netpol- network policies
No = nodes	ns= namespaces	pdb= pod
po= pods	Pod preset	Pod templates
Psp= pod security policies	Pv= persistent volumes	pvc= persistent volume claims
quota= resource quotas	rc= replication controllers	Role bindings
roles	rs= replica sets	sa=service account
sc= storage classes	secrets	sts= stateful sets

• Cluster Introspection

COMMANDS	FUNCTION
Get version information	Kubectl version
Get cluster information	Kubectl cluster-info
Get the configuration	Kubectl config g view
Output info about a node	Kubectl describe node<node>

Other Quick Commands

Launch a pod with a name an image : Kubectl run<name> --image=<image-name>

Create a service in <manifest.yaml> : Kubectl create -f <manifest.yaml>

Scale replication counter to count the number of instances : Kubectl scale --replicas=<count>

Map external port to internal replication port : Expose rc<name> --port=<external>--target-port=<internal>

To stop all pod in <n> : Kubectl drain<n>-- delete-local-data--force--ignore-daemonset

Allow master nodes to run pods : Kubectltaintnodes --all-noderole.kuernetes.io/master-

IntelliPaat

Maven cheat sheet

Getting started with Maven

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
```

Create archetype from existing project

```
mvn archetype:create-from-project
```

Main phases

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

Useful command line options

-DskipTests=true compiles the tests, but skips running them

-Dmaven.test.skip=true skips compiling the tests and does not run them

-T - number of threads:
-T 4 is a decent default
-T 2C - 2 threads per CPU

-rf, --resume-from resume build from the specified project

-pl, --projects makes Maven build only specified modules and not the whole project

-am, --also-make makes Maven figure out what modules out target depends on and build them too

-o, --offline work offline

-X, --debug enable debug output

-P, --activate-profiles comma-delimited list of profiles to activate

-U, --update-snapshots forces a check for updated dependencies on remote repositories

-ff, --fail-fast stop at first failure

Essential plugins

Help plugin — used to get relative information about a project or the system.
mvn help:describe describes the attributes of a plugin
mvn help:effective-pom displays the effective POM as an XML for the current build, with the active profiles factored in.

Dependency plugin — provides the capability to manipulate artifacts.
mvn dependency:analyze analyzes the dependencies of this project
mvn dependency:tree prints a tree of dependencies

Compiler plugin — compiles your java code.
Set language level with the following configuration:

```
<plugin>
 <groupId>org.apache.maven.plugins</groupId>
 <artifactId>maven-compiler-plugin</
  artifactId>
 <version>3.6.1</version>
 <configuration>
   <source>1.8</source>
   <target>1.8</target>
 </configuration>
</plugin>
```

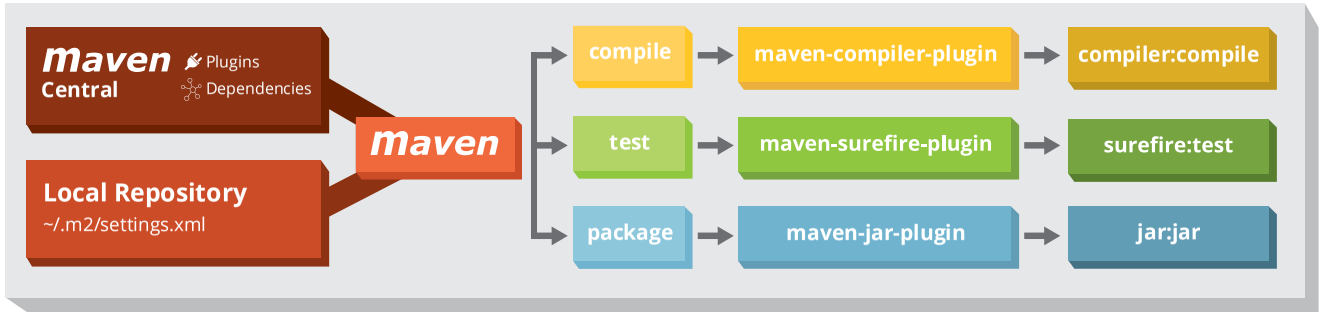
Version plugin — used when you want to manage the versions of artifacts in a project's POM.

Wrapper plugin — an easy way to ensure a user of your Maven build has everything that is necessary.

Spring Boot plugin — compiles your Spring Boot app, build an executable fat jar.

Exec — amazing general purpose plugin, can run arbitrary commands :)

The big picture



Spring Boot and Web annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses @Configuration, @EnableAutoConfiguration and @ComponentScan.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

T **@Controller** - marks the class as web controller, capable of handling the requests. T **@RestController** - a convenience annotation of a @Controller and @ResponseBody.

M T **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.

Spring Cloud annotations

Make you application work well in the cloud.

T **@EnableConfigServer** - turns your application into a server other apps can get their configuration from.

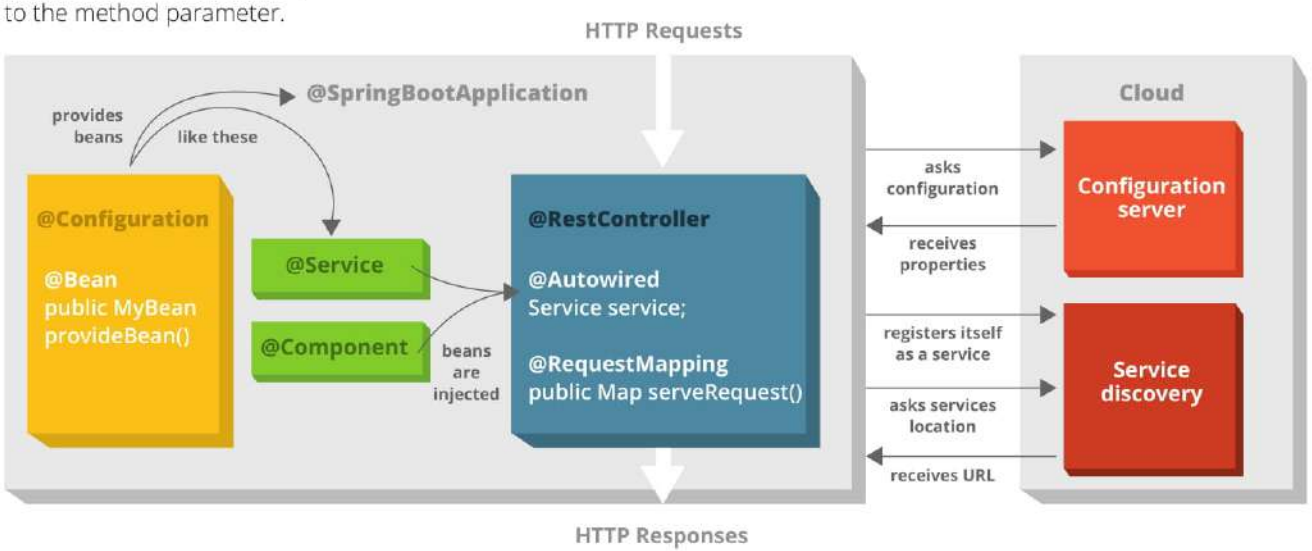
Use `spring.application.cloud.config.uri` in the client @SpringBootApplication to point to the config server.

T **@EnableEurekaServer** - makes your app an Eureka discovery service, other apps can locate services through it.

T **@EnableDiscoveryClient** - makes your app register in the service discovery server and discover other services through it.

T **@EnableCircuitBreaker** - configures Hystrix circuit breaker protocols.

M **@HystrixCommand(fallbackMethod = "fallbackMethodName")** - marks methods to fall back to another method if they cannot succeed normally.



Spring Framework annotations

Spring uses dependency injection to configure and bind your application together.

T **@ComponentScan** - make Spring scan the package for the @Configuration classes.

T **@Configuration** - mark a class as a source of bean definitions.

M **@Bean** - indicates that a method produces a bean to be managed by the Spring container.

T **@Component** - turns the class into a Spring bean at the auto-scan time. T **@Service** - specialization of the @Component, has no encapsulated state.

C F M **@Autowired** - Spring's dependency injection wires an appropriate bean into the marked class member.

T M **@Lazy** - makes @Bean or @Component be initialized on demand rather than eagerly.

C F M **@Qualifier** - filters what beans should be used to @Autowire a field or parameter.

C F M **@Value** - indicates a default value expression for the field or parameter, typically something like `"#{systemProperties.myProp}"`

C F M **@Required** - fail the configuration, if the dependency cannot be injected.

Legend

- T - class
- F - field annotation
- C - constructor annotation
- M - method
- P - parameter