Change - detection

In Angular (zone.js used), whole component tree is wrapped by zone.

Any change triggered due to some event (event binding set on that component) triggers change-detection cycle for entire tree. Each property & event binding checked for changes & accordingly DOM updated.

↓

All template-bindings.

(event, 2-way-binding, string interpolation, property)

DFS traversal employed for change-detection.

Performance-intensive work should be avoided from template-binding getters fn. should not perform super-complex tasks.

During change-detection, Angular performs it twice.

(components visited twice)

(Happens only during development, not in production)

Twice in development mode - to detect unwanted value changes.

(will throw error if encountered different values in both cycles)

optimize change-detection

(1) Not put expensive (performance-intensive) calculations/expressions in template-bindings.
simple - straightforward ✓
complex calculations ✗
calling functions (except event-bindings & signals) ✗
(avoid calling fn. except for event-bindings & signals ≠ reads)
(getters should be used. they should do basic thing as well)

(2) consider telling Angular if a certain event doesn't matter for change-detection (Avoid zone pollution)
(console.log() in ngOnInit() could trigger change-detection, so remove such statements)
→ set via setTimeOut fn. (timer-expiring event kick change-detection)
you can opt-out of zone.js watch-mode

```
private zone = inject(NgZone)

this.zone.runOutsideAngular(() => {
  setTimeout(() => {
    console.log('timer expired');
  }, 5000);
});
```

This makes the callback fn. run outside Angular change detection f/ or outside zone.js watch mode

③ use on-push strategy

Opt-in strategy can be enabled per-component basis. This makes change detection run potentially less often for the given component.

The component on which on-push strategy is used will only change because some event occured inside of sub-component tree or because an input value changed in the component.

We can trigger change-detection manually.

So on-push triggers change-detection in

1. Manual trigger of CD
2. Input property value changed
3. Event occured anywhere in the component or sub-components of this component

Hence, it limits the amount of events/things that could trigger change-detection

signal - changes also lead to onPush becoming active (lead to component to be checked again & child components also)

If you are not using signals in your service to communicate data, it could lead to problems with components that use the service & have onPush strategy enabled.

such components won't get the latest data from the service.

you need to manually trigger the change-detection.

private cdRef = inject(ChangeDetectorRef);

use the changeDetectorRef token in component where you wish to trigger change-detection.

To get notified of changes for change-detection to be triggered 'rxjs' is used.

service.ts -

import {BehaviorSubject} from 'rxjs'

export class MessageService {
  messages$ = new BehaviorSubject<string[]>([]);
  addMessage(message: string) {
    this.messages = [...this.messages, message];
    this.messages$.next(this.messages);
  }
}

emit the value

BehaviorSubject through "next" method carries updated data.

In component using the service, subscribe

```
export class MessagesListComponent implements
  OnInit {

ngOnInit() {
  this.messagesService.messages$.subscribe(
    (messages) => { this.messages = messages;
      this.cdRef.markForCheck();
  });
```

So for Manually triggering changes :-

1. Set BehaviorSubject in service.
2. Do BehaviorSubject.next(changed-property).
3. In component use ChangeDetectorRef token. (inject) & onInit, subscribe to BehaviorSubject property from service & in subscription markForCheck() for change-detection to trigger.

A shortcut to this situation exists —

You can use async pipe in component template.
(Angular sets-up a subscription behind the scenes)

unsubscription also taken care by Angular.
The async pipe will also trigger change detection whenever a new value for the subject is recieved.

So. Template

```
<ul>
  @for (message of messages$ | async; track message)
  { <li>{{ message }}</li>
  }
</ul>
```

component. class

```
import {AsyncPipe} from '@angular/common';

export class
@Component ({
    imports: [AsyncPipe],
})
export class MessagesListComponent{
  ---
    messages$ = this.messagesService.messages$
```

If you have only event-bindings & signals used in entire application we can get ride of zone.js. We can trigger manual change detection if needed.

Expiring Timers like fn. won't trigger change-detection on its own though (like in zone.js). If you use signals & update them, then

change-detection can kick-in.

Getting rid of zone.js leads to -

1. More fine-grained change detection
2. smaller bundle size
3. less behind the scenes check